UNIVERSITA' DEGLI STUDI
DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE

Honeywell Bull
CENTRO STUDI
INFORMATICA
E AUTOMAZIONE

NOTE DI SOFTWARE

ARCHIVIO
STORICO
ASSOCIAZIONE
POZZO DI MIELE
FPDM-120
RNSW-127

ANNO 1988
39/40
MARZO-GIUGNO

Note di software è una pubblicazione trimestrale, edita a cura del Dipartimento di Scienze dell'Informazione dell'Università di Milano e dal Centro Studi Informatica e Automazione della Honeywell Bull Italia.

Note di software intende costituire uno strumento per la circolazione di informazioni, idee, metodologie ed esperienze nell'area delle tecnologie e della produzione del software.

La collaborazione a **Note di software** è libera. Chi desiderasse pubblicare un contributo è pregato di prendere contatto con il Direttore responsabile o con il Comitato di Redazione. In particolare si sollecitano contributi nella forma di monografie, da inviare in triplice copia, e che non dovrebbero superare le cinquemila parole.
I lavori ricevuti per la pubblicazione verranno revisionati dai membri del Comitato di Revisione Scientifica, che possono avvalersi della collaborazione di specialisti del settore. I lavori pubblicati riflettono comunque le opinioni dei loro autori.

Note di software viene distribuito ad Aziende, Enti e specialisti del settore che ne facciano richiesta alla Redazione.

Direttore Responsabile: Paolo Ghelfi
Honeywell Bull Italia S.p.A., via ai Laboratori, Pregnana Milanese

Comitato di Redazione: Stefania Bandini, Maria Paola Bonacina, Francesco Gardin, Roberto Polillo
Università di Milano
Dipartimento di Scienze dell'Informazione
via Moretto da Brescia, 9 - 20133 Milano
tel. (02) 2772233

Comitato di Revisione Scientifica: Alberto Bertoni, Gianluigi Castelli, Giovanni Degli Antoni, Giorgio De Michelis, Mario Italiani, Gaetano Aurelio Lanzarone, Giancarlo Martella, Marco Maiocchi, Giancarlo Mauri.

Autorizzazione del Tribunale di
Milano n. 87 del 23 febbraio 1977

UNIVERSITA' DEGLI STUDI
DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE

FPDM-120

Honeywell Bull
CENTRO STUDI
INFORMATICA
E AUTOMAZIONE

QUESTO NUMERO 1

— RASSEGNA DEGLI IPERTESTI
di G.E. Caja, P. Paiardi, C. Testero 3

— UN ESPERIMENTO CONNESSIONISTA SULLA
COMPRENSIONE DEL LINGUAGGIO NATURALE
di S. Nolfi, D. Parisi 22

— CAESAR: COOPERATING AGENTS FOR EXPERT
SYSTEM ACTIVITY REPRESENTATION
di A. D'Angeli, E. Pagello 25

— LA BASE DI CONOSCENZA DI CHAOS:
LA STRUTTURA ED IL LESSICO DELL'UFFICIO
di A. Gerevini, S. Bottarelli 35

— ADVANCED PROGRAM GENERATION TECHNIQUES
BOOTSTRAPPING AND PARTIAL EVALUATION
di M.P. Bonacina 50

— ARTIFICIAL INTELLIGENCE AND CREATIVITY:
A PROPOSAL FOR AN AUTOMATION INVENTING NEW OBJECTS
di R. Pizzi, D. Galli 56

— THE INTERTEST INITIATIVE WITHIN OSIRIDE PROJECT
di L. Lenzini 66

IL NOTIZIARIO SUI CD-ROM

— NOVITÀ DA SEATTLE 80

— LE ULTIME NOVITÀ SULLE MEMORIE OTTICHE 80

RECENSIONI

— a cura di S. Bandini 82

QUESTO NUMERO

Questo doppio numero di Note di Software presenta una interessante selezione di contributi su svariati settori di interesse per la ricerca in informatica e la produzione del sofware: ipertesi, linguaggio naturale, sistemi esperti, automazione d'ufficio, generazione automatica di programmi, automazione della creatività e problemi di standard.

Gli ipertesi sono uno strumento per l'organizzazione del lavoro intellettuale su calcolatore che sta riscuotendo notevole interesse in tutti gli ambiti di ricerca e non, specialmente quelli più interessati ai problemi dell'interfaccia uomo-macchina. Questo numero di Note di Software apre con una rassegna sugli ipertesi realizzata da G.E. Caja, P. Pirardi e C. Testero. La rassegna presenta sia le caratteristiche generali degli ipertesi sia le particolari applicazioni de alcuni ipertesi in commercio.

L'interfaccia uomo-macchina é l'obbiettivo anche del lavoro di S. Nolfi e D. Parisi, un originale contributo al classico problema della comprensione del linguaggio naturale.

Seguono due articoli nell'ambito dei sistemi esperti e dell'automazione d'ufficio, rispettivamente di A. D'Angelo e E. Pagello e di A. Gerevini e S. Bottarelli. Quest'ultimo presenta lo stato della ricerca sul sistema CHAOS, attuamente in corso presso il Dipartimento di Scienze dell'Informazione di Milano.

Il lavoro di M. P. Bonacina presenta e confronta due tecniche di generazione di programmi. La prima é la ben nota tecnica bootstrap che é stata ormai applicata con successo alla produzione di compilatori. La seconda si chiama valutazione parziale ed é considerata in ambito di ricerca uno strumento emergente per la produzione del software.

R. Pizzi e D. Galli presentano poi una piacevole escursione nel mondo della creatività, che rappresenta forse la barriera più ostica per tutti coloro che sognano di costruire macchine che riproducano approssimazioni del comportamento umano.

Il numero é concluso dal contributo di L. Lenzini, che tratta del progetto OSIRIDE, un insieme uniforme di servizi informatici per la comunità scientifica italiana messo a disposizione dal Consiglio Nazionale delle Ricerche (CNR).

La REDAZIONE

RASSEGNA SUGLI IPERTESTI

G. E. Caja, P. Piardi, C. Testero

Dipartimento di Scienze dell'Informazione

Università degli Studi di Milano

RIASSUNTO

Gli ipertesti aggiungono nuove funzionalità ai documenti elettronici. Il presente lavoro offre una descrizione generale degli ipertesti, di come operano, delle loro proprietà e funzionalità. Inoltre è presentato come esempio un modello matematico. Segue una panoramica sui vari campi di applicazione e, per ognuno di essi, una descrizione dei prodotti attualmente in commercio.

ABSTRACT

Hypertexts add new functionalities to electronic documents. This report shows a general description of Hypertext Systems, their properties, functionalities, and how they operate. As an example, a mathematical model is also added. Hence, a view of applications range and a brief description of products actually realized follow.

1. INTRODUZIONE

Nel 1945 nasceva per la prima volta con Vannevar Bush, presidente della Roosevelt's Scienze Advisor, l'esigenza di estendere la tradizionale nozione di testi lineari per permettere più complesse organizzazioni dell'informazione.

Nell'articolo "As we way may think" (1945) Bush suggeriva di prendere come modello i processi mentali umani associativi. Il processo di pensiero non costruisce nuove idee una alla volta, ma piuttosto le sviluppa in parallelo, a differenti livelli; ciascuna idea contribuisce alla creazione di nuove idee o dipende dalle altre.

Dall'articolo "As we may think":

> "La mente unama ... opera associativamente. L'uomo non può sperare di simulare pienamente questo processo mentale artificialmente, ma deve essere in grado di imparare da esso. Non si può sperare di eguagliare la velocità e flessibilità con cui la mente segue un cammino associativo, ma deve essere possibile avvicinarsi alla chiarezza e durevolezza degli elementi recuperati dalla memoria ...".

Basandosi su questo modello cognitivo Ted Nelson, descrivando il sistema *Xanadu,* definì nel 1960 la nozione di ipertesto.

Egli definì un ipertesto come:

> " ... combinazione di un testo in linguaggio naturale con la capacità dei computers di seguire interattivamente diverse ramificazioni ... di un testo non lineare ... che non può essere stampato convenientemente seguendo una convenzionale impaginazione".

La nuova tecnologia rende tale concetto effettivamente realizzabile, aprendo nuove vie per l'utilizzazione del computer come mezzo di comunicazione e di pensiero.

Un ipertesto è essenzialmente un testo non lineare, costituito da documenti contenenti una collezione di nodi collegati da links diretti. Un nodo rappresenta semplicemente una parte di testo, mentre i links tra i nodi danno all'ipertesto il suo aspetto non lineare. Il contenuto dei nodi non è necessariamente un testo, ma può essere un disegno, suoni codificati o anche animazione.

L'insieme di tutti i documenti di un sistema Hypertext viene definito iperdocumento.

2. HYPERTEXT SYSTEMS

2.1 Proprietà

2.1.1 I nodi

Anche se l'essenza di un ipertesto è espressa dal tipo di linking che la macchina è in grado di supportare, i nodi contribuiscono in modo significativo alla definizione delle operazioni che un sistema ipertesto può fornire.

Il contenuto dei nodi può essere di diverso tipo, non è quindi limitato a testi. I nodi possono rappresentare immagini grafiche, combinazioni di testo e grafica, suoni digitalizzati, e anche animazione.

Certi sistemi ipertesti permettono che i nodi siano di tipi diversi. I sistemi che usano nodi tipizzati generalmente offrono un colore particolare, ampiezza, o icone, per ciascun tipo di nodo (in particolare Note Cards, Intermedia, Ibis).

Per alcune applicazioni sta crescendo l'interesse verso nodi "semistrutturati", cioè nodi tipizzati che contengono campi etichettati e spazi per i rispettivi valori.

Un altro meccanismo per aggregare una informaizone correlata in un ipertesto è il "composite node". Molti nodi tra loro in relazione sono messi insieme e la collezione è trattata come un singolo nodo, con il proprio nome, tipo, versione etc..

Per quanto riguarda la struttura dei nodi, c'è discordanza tra i sistemi che preferiscono tenere molti piccoli nodi, in modo da massimizzare il grado di organizzazione dell'informazione, e quelli che preferiscono tenere solamente pochi nodi di notevoli dimensioni per minimizzare il ritardo e la complessità di attraversamento dei nodi.

Ci sono due modi per implementare i nodi in un file system:

— ogni nodo è un file

— tutti i nodi sono memorizzati in un grande file.

Il secondo metodo ha lo svantaggio che non ci possono essere molti utenti concorrenti senza un servizio più accurato di protezione del file system.

2.1.2 I links

I links possono connettere:

— il punto in cui un documento ha un riferimento con un'altra parte dello stesso documento;

— commenti ed annotazioni con il testo al quale si riferiscono;

— due successive parti di testo, oppure una parte con tutti i suoi successori;

— accessi ad una tabella, oppure immagini ad una descrizione più dettagliata o ad altre tabelle o ad altre figure.

I links possono essere definiti da nomi, essere di più tipi e possedere un ricco insieme di proprietà. Inoltre, in alcuni sistemi è possibile rendere attiva o meno la loro visualizzazione, così che, in caso di disattivazione, il documento appare come un testo comune, lineare.

L'introduzione dei links in un Text system implica l'aggiunta di un insieme di meccanismi per la loro gestione, ossia per crearli, rimuoverli, modificarli nel nome o negli attributi etc..

Ci sono due metodi per collegare due punti in un Hypertext: referential links, organizational links.

Referential links

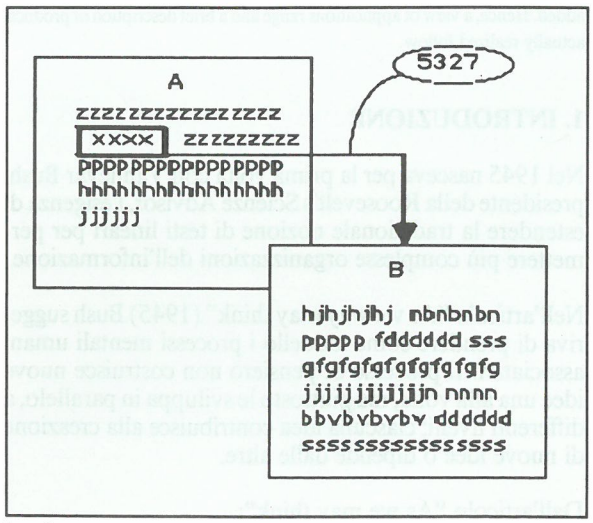
Il metodo che utilizza "referential links" organizza il testo secondo una struttura non gerarchica, a rete.

I links possono unire punti o regioni in un testo, ed hanno due estremità: la sorgente, che agisce come "reference", e la destinazione, che agisce come "referent".

Un "punto" (sorgente/destinazione) è una icona, che indica la presenza del link, mostrandone il nome.

Una "regione" è un insieme di caratteri contigui visualizzato come singola unità (vedi fig. 1). Il modo più semplice di implementare le regioni è quello di rappresentarle come un intero nodo. Il problema è come visualizzare la regione all'utente; essa può essere segnalata, ad esempio in reverse o tramite colori diversi. La difficoltà consiste nell'evidenziare le regioni che vengono a sovrapporsi, in modo da mantenere la gerarchia.

Due potrebbero essere le soluzioni: visualizzare soltanto una singola regione alla volta, oppure usare un certo schema di colori. Le regioni possono essere difficili da manipolare, in quanto è necessario fornire un sistema di funzioni per copiare, modificare, cancellare regioni e sottostringhe.



La figura mostra un link tra un punto come sorgente ed una regione come destinazione.

Organizational links

Il metodo che utilizza gli "organizational links" organizza il documento secondo una struttura gerarchica; i links connettono un nodo genitore con tutti i suoi figli, formando un albero all'interno del ipergrafo. Essi corrispondono ai links 'is-a' della teoria delle **reti semantiche**.

Invece di apparire come espliciti tokens in ciascun nodo, gli organizational links sono talvolta attraversati da un meccanismo separato che utilizza comandi speciali come "goto-parent", "goto-first", "goto-child", etc..

In altri casi i nodi stessi (ad esempio *"toc"* in Textnet e *"fileboxes"* in Notecards) rappresentano la struttura organizzazionale.

Alcuni sistemi Hypertext forniscono soltanto una struttura gerarchica (ad es. *NLS, AUGMENT, ZOG, OUTLINE PROCESSORS*), altri invece solamente una struttura non gerarchica (ad es. *XANADU, NEPTUNE*), ed infine altri entrambi i tipi (ad es. *TEXTNET, NOTECARDS*). Un vantaggio dei sistemi orientati strettamente ad una struttura gerarchica ad albero è quello di avere un liguaggio molto semplice di comandi per l'attraversamento: da un nodo l'utente può spostarsi solo verso un genitore, un fratello, o un figlio. Questa semplicità può aiutare a risolvere il problema del *disorientamento dell'utente.*

Naturalmente uno svantaggio di tale struttura è il fatto che l'utente, che crea una organizzazione gerarchica, deve anticipare i più importanti criteri per i futuri accessi all'informazione.

Keywords links

Esiste un altro tipo di linking, implicito, che utilizza keywords per la ricerca di un documento o di una parte di esso. A differenza del metodo esplicito, che connette un nodo con un altro, la ricerca per keywords collega un insieme di nodi.

2.1.3 Meccanismi di attraversamento

Un ipertesto è attraversato tramite l'attivazione dei suoi links. L'utente sceglie, secondo le proprie esigenze, un particolare cammino dell'ipergrafo.

Molti sistemi, come Zog, Augment, Neptune, forniscono uno speciale supporto alla costruzione e visione di documenti con struttura gerarchica.

Il lettore di un ipertesto, seguendo link dopo link nella lettura di porzioni del documento ha l'esigenza di tenere traccia del cammino seguito, per potere eventualmente tornare nella posizione di partenza. È compito del sistema fornire adeguati meccanismi di backtracking.

2.1.4 Interfaccia utente

Un ipertesto è visitato interattivamente. Quando un lettore visualizza un nodo, i links presenti possono essere seguiti o no a discrezione del lettore; se un link viene seguito, allora il nodo destinazione è reso visibile in modo che possa a sua volta essere letto. In un display a più finestre, l'operazione di seguire un link e di vedere ciò a cui punta è semplice da implementare con un mouse. *Intermedia* e *NoteCards* sono esempi particolarmente validi di una innovativa tecnologia per l'implementazione dell'interfaccia utente.

2.1.5 Multiutenza

Il primo sistema che introdusse il concetto di editing distribuito a più utenti è stato NLS, poi chiamato Augment.

Molte persone possono accedere ad un iperdocumento simultaneamente, e l'iperdocumento stesso può essere distribuito su una rete di workstations.

Un'implementazione di ipertesto, quindi, deve considerare i problemi di controllo della concorrenza e di salvataggio.

2.1.5.1 Contesti

Delisle e Schwartz hanno suggerito l'uso di contesti per partizionare gli ipertesti. Il problema che porta alla necessità di utilizzare il concetto di contesto nasce dall'esigenza di regolare l'accesso ad un insieme di informazioni a più utenti. Ciascun utente può "derivare" una vista privata del grafo e fare delle modifiche entro questa vista locale. Quando un insieme di cambiamenti è completato, esso viene reso visibile ad altri membri di un progetto comune tramite un'operazione di "merging" della vista privata con una vista comune detta "master". Possono nascere dei conflitti se altri autori hanno modificato il master dal momento della creazione della vista privata. È quindi necessario individuare questi conflitti. Da questa nozione di "derive-merge" si arriva alla nozione di contesti.

Ci sono due fondamentali modelli:

1 - *derive model:* ciascun contesto è subordinato ad un singolo contesto genitore. Quando viene creato il nuovo contesto, tutti i nodi e i links sono visibili nel nuovo contesto subordinato. Il contesto figlio può essere usato come spazio di lavoro privato; ogni modifica è visibile soltanto entro il contesto figlio. In seguito i cambiamenti possono essere riportati nel contesto padre.

2 - *merge model:* il nuovo contesto creato risulta vuoto, Una collezione di nodi e links possono essere mossi nel nuovo contesto da un qualsiasi altro contetso esistente tramite un'operazione di merge.

Il derive model può essere visto come un caso del più generale merge model in cui vengono messi certi limiti sull'operazione di merge.

È possibile dare una definizione di contesto, secondo il formalismo nelle sezioni precedenti.

2.2 CD-ROM e Ipertsti

CD-ROM ed Ipertesti non sono la stessa cosa, non sono neanche tecnologicamente interdipendenti, nonostante ciò vengono spesso considerati assieme. Entrambi sono "media" di informazione:

1 - CD-ROM è hardware che mantiene delle informazioni;

2 - l'Ipertesto è software che definisce come è strutturata l'informazione.

I dati sui CD-ROMs possono essere strutturati in tutti modi possibili, analogamente a come sono strutturati su un hard disk, quindi anche come ipertesto. Due porprietà dei CD-ROMs sono particolarmente utili per gli ipertesti: la grande capacità e la natura statica read-only.

La capacità dei CD-ROMs permette di memorizzare un grande ammontare di informazioni, e un ipertesto è proprio un modo per rendere accessibile all'utente finale questa grande quantità di dati.

Inoltre il problema dello sviluppatore di un ipertesto è quello di anticipare e costruire i cammini più utili all'interno dell'ipergrafo, dato che è praticamente impossibile considerare tutti i possibili cammini. La grande capacità dei CD-ROMs rende possibilie la memorizzazione di un grande numero di cammini possibili, anticipando così le necessità di molti utenti.

L'ipertesto su CD-ROM è essenzialmente differente dall'ipertesto dinamico, inteso come tool per una scrittura cooperativa o come ambiente interattivo di comunicazione tra lettore e scrittore. L'ipertesto su CD-ROM è un mezzo di comunicazione della conoscenza umana, in assenza però di feedback dal lettore allo scrittore.

Gli ipertesti dinamici, modificabili, hanno però un problema per quanto riguarda le modifiche. I links hanno origine in un documento e, a seconda delle implementazioni, possono puntare ad un intero documento o ad una parte di esso. Quindi il nome del documento destinazione è modificato, i links possono essere cambiati. Questo è di notevole difficoltà per grafi contenti decine di migliaia di nodi.

Su CD-ROM il documento base non può essere alterato o rimosso. Annotazioni o nuovi nodi possono essere aggiunti, ma una struttura di base, non modificabile sarà sempre presente.

Il problema principale che si pone nell'utilizzo di CD-ROM per l'editoria elettronica è la perdita di alcune proprietà desiderabili del mezzo scritto.

Un libro scritto può essere sottolineato, visualizzando parti importanti per il lettore, e possono essere inseriti segnalibri per aiutare il lettore a ritrovare in un momento successivo un punto del libro. Quindi per simulare la lettura di un libro è richiesta una certa forma di revisione dei documenti. Per quanto riguarda i CD-ROMs, sappiamo che non sono modificabili, ma è possibile inserire le note personali dell'utente in un file associato al documento in esame.

Un'altra proprietà dei libri stampati è la possibilità di sfogliarli, offrendo al lettore una visione rapida e generale del contenuto; ad esempio le figure sono un utile richiamo all'attenzione dell'utente. È quindi opportuno memorizzare una lista di tutte le figure e presentare tale lista in modo rapido al lettore, il quale, selezionandone una, può risalire al documento desiderato.

3. LIMITAZIONI DEI SISTEMI ESISTENTI

Si possono individuare due classi di problemi inerenti agli Ipertesti: problemi che riguardano le correnti implementazioni e problemi che sembrano essere insiti nel concetto stesso di Ipertesto.

La prima classe di problemi include ritardi nella visualizzazione del materiale a cui si riferisce un link, restrizioni sui nomi e proprietà dei links, mancanza o limitazioni nei browsers, etc..

Alla seconda classe appartengono invece due problemi fondamentali, che possono effettivamente limitare l'utilità dell'Ipertesto: disorientamento e costo cognitivo.

L'Ipertesto è un mezzo di informazione che permette di organizzare materiale in un modo molto più complesso rispetto ai tradizionali testi lineari. A causa di tale complessità, l'utente ha necessità di sapere, in ogni momento, dove si trova e come raggiungere un altro punto della rete.

Naturalmente questo problema di disorientamento è presente anche nei tradizionali documenti organizzati linearmente ma, in questo caso, il lettore ha soltanto due possibilità: egli può cercare il testo desiderato o più indietro o più avanti rispetto al punto in cui si trova. Gli Ipertesti offrono più gradi di libertà, più dimensioni in cui muoversi e quindi più possibilità per l'utente di perdersi. Le due principali soluzioni tecnologiche per diminuire il disorientamento sono browsers grafici e meccanismi di interrogazione/ricerca. I browsers permettono di visualizzare nodi e links in uno spazio 2-3 dimensionale, offrendo proprietà utili per una chiara e visibile differenziazione (es.: colore, ampiezza, etc.) e mantenendo certe similitudini con il nostro ambiente fisico (es.: due oggetti distinti non possono occupare lo stesso spazion e non possono cambiare posizione se non sono mossi dall'utente).

Tuttavia non c'è nessuna precisa topologia per la rappresentazione dello spazio di informazione, cosicché l'utente, fino a quando non acquisisce una approfondita conoscenza delle informazioni contenute nell'iperdocumento, è, per definizione, disorientato.

Ci sono inoltre altre difficoltà, nel rappresentare il grafo, che sorgono quando sono presenti un grande numero di

nodi o di links, oppure vi sono frequenti cambiamenti nella rete. Una soluzione può essere quella di applicare al database tecniche di ricerca o di interrogazione per localizzare i nodi, utilizzando keywords e predicati logici su attributi di nodi e links.

Un altro problema fondamentale nell'uso di Ipertesti è la difficoltà ad accostarsi all'addizionale costo mentale richiesto per creare, dare nomi appropriati e tenere traccia di links. Tale problema è chiamato costo cognitivo.

Inoltre, nel processo di lettura, l'utente si trova a dover decidere tra un grande numero di scelte riguardo a quali links seguire. La presenza di tali scelte comporta un certo sforzo mentale, che non è richiesto nel caso dei testi lineari in cui l'autore ha già definito l'iter da seguire nella lettura del documento.

4. MODELLO MATEMATICO

L'informazione può essere memorizzata e recuperata secondo diversi livelli di dettaglio e differenti punti di vista. Questo è ciò che in generale si intende per 'astrazione'. È quindi particolarmente opportuno studiare e valutare i diversi meccanismi di astrazione per gli ipertesti. Vediamo una formalizzazione matematica del concetto di ipertesto.

DEFINIZIONE 1 - Un ipertesto η è un insieme costituito da:

1 - Un insieme di oggetti primitivi, P', e di oggetti di informazione, I': $P' \cap I' = 0$;

2 - Un insieme di predicati, p;

3 - Un insieme di attributi (o proprietà), A.

Dato un ipertesto η, $p[\eta]$ rappresenta l'insieme di oggetti primitivi di η, $I'[\eta]$ è l'insieme di oggetti di informazione di η, $A[\eta]$ è l'insieme di predicati di η.

4.1 Predicati

P è un insieme di predicati, costituito da tre insiemi:

1 - Un insieme, π_1, di predicati ad un argomento che caratterizzano gli oggetti.

2 - Un insieme, π_2, di predicati a due argomenti tali che se P(X,Y) è vero, allora esiste la relazione P tra i due oggetti X e Y; se invece è falso, allora tale relazione non esiste.

3 - Un insieme, π_3, contenente l'elemento 'Property' tale che, se Property (X,Y,Z) è vero, allora Z è il valore della proprietà Y dell'oggetto X.

4.2 Oggetti e loro istanze

L'insieme $O = P' \cup I'$ è chiamato insieme di oggetti. In seguito, parlando di oggetti di informazione si intenderà trattare di oggetti di tipo testuale.

Gli oggetti di informazione possono esere messi in relazione con gli oggetti primitivi tramite il predicati INSTANCEof e la funzione INSTANCES. INSTANCEof(X,Y) significa che X è un oggetto di informazione che è una istanza dell'oggetto primitivo Y. INSTANCES(X) = { X_1, X_2, ... } significa INSTANCEof(X_1, X), INSTANCEof(X_2, X), ...

DEFINIZIONE 1 - Un ipertesto è strettamente definito se e solo se

$$\forall X((X \in I') \Rightarrow (\exists Y(\text{INSTANCEof}(X, Y))))$$

Tutti gli oggetti di informazione hanno così definiti i propri oggetti primitivi. Analogamente, tutti i nodi contenenti informazione sono scelti da un insieme di pre-esistenti nodi primitivi, e l'utente dell'ipertesto non può creare un n uovo nodo senza che prima sia stato creato un nodo primitivo della stessa categoria.

ASSIOMA 1 - Un oggetto di informazione condivide gli attributi del corrispondente oggetto primitivo.

Il contenuto di un oggetto di informazione è rappresentato dalla proprietà INFO. Il valore di tale proprietà è un insieme contenente coppie (posizione, valore), ove "posizione" è un numero reale che identifica la posizione del "valore" nell'oggetto di informazione.

Ad esempio, consideriamo il seguente oggetto di informazione:

1 - Un quadrato è una figura
2 - : geometrica
3 - che è un
4 - : rettangolo
5 - { puntatore all'oggetto contenente l'informazione sul rettangolo }
6 - con tutti i lati uguali.

Per l'oggetto di informazione SQUARE abbiamo che:

members (square-info) = { (1, "Un quadrato è una figura"), (2, : geometrica), ... , (4, : rettangolo), ... }

Le parole chiave sono marcate nel testo per mezzo di ":".

DEFINIZIONE 2 - Una istanza della tupla (posizione, valore) è chiamata *linea*

DEFINIZIONE 3 - Se S è un insieme di linee, allora posizione [s] è la proiezione del campo posizione delle linee in S.

Se S è un insieme di linee, allora valore [s] è la proiezione del campo valore delle linee in S.

4.3 Meccanismi di astrazione

4.3.1 Aggregazioni

L'aggregazione è un meccanismo di astrazione che per-

mette di riferirsi ad una collezione di oggetti tramite un unico identificatore.

Gli oggetti aggregati sono sequenze di oggetti che rispettano speciali assiomi.

Tutte le operazioni definite per un oggetto aggregato sono lecite anche per tutti i sottosistemi di tale aggregato. Inoltre le aggregazioni devono soddisfare i seguenti vincoli:

ASSIOMA 1 - Gli elementi di un oggetto aggregato sono unici.

Gli oggetti aggregati devono corrispondere ad una certa astrazione del mondo reale e quindi la stessa collezione di oggetti non può essere vista come differenti aggregazioni (questo per mantenere l'integrità dell'ipertesto).

ASSIOMA 2 - Se un oggetto aggregato è un oggetto primitivo, allora tutte le componenti dell'aggregazione devono essere oggetti primitivi.

ASSIOMA 3 - L'istanza di un oggetto aggregato è formata dalle istanze degli oggetti componenti.

È da notare che un oggetto aggregato riassume la nozione di n-uple dei database relazionali. Quindi è possibile definire proiezioni, unioni, selezioni, etc. anche per le aggregazioni.

4.3.2 Generalizzazioni

Una generalizzazione è un meccanismo di astrazione che permette di riferirsi ad una collezione di oggetti tramite un oggetto generico che riassume le caratteristiche comuni degli oggetti. È da notare che la generalizzazione, diversamente dal meccanismo di aggregazione, fa perdere agli oggetti individuali la proprie individualità. Gli oggetti generali condividono le proprietà degli oggetti che generalizzano.

Una forma di generalizzazione risulta chiara nella definizione della funzione ISTANCES. Essa è uno pseciale tipo di generalizzazione ove un oggetto primitivo è un oggetto generale che ha come oggetti specializzati le proprie istanze.

Le geneneralizzazioni sono definite da una funzione da O a O, tale che:

se GENERALIZE(X) = Y allora Y è la generalizzazione di X.

ASSIOMA 1 - La relazione GENERALIZE è assimetrica, non riflessiva e transitiva.

ASSIOMA 2 - Se X è un oggetto di informazione, istanza dell'oggetto Y, allora Y è una generalizzazione di X.

(ISTANCEof(X, Y) ⇒ (GENERALIZE(X) = Y))

ASSIOMA 3 - Se Y è una generalizzazione di X, e X è un oggetto di informazione e Y è un oggetto primitivo, allora X è un insieme di istanze di Y.

$$[[(\text{GENERALIZE}(X) = Y) \text{ AND } (X \in I') \text{ AND } (Y \in P')] \Rightarrow (X \in \text{INSTANCES}(X))]$$

TEOREMA 1 - Se X è una generalizzazione di Y e Y_m è nell'insieme di istanze di Y, allora Y_m è nell'insieme di istanze di X.

DEFINIZIONE 1 - (Ordinamento parziale) Una coppie ordinata (S, R), ove S è un insieme a R è una relazione da S a S, è un ordinamento parziale se:

1 - R è asimmetrica
2 - R è riflessiva
3 - R è transitiva.

DEFINIZIONE 2 - (Coprire) Dato un ordinamento parziale (S,R), un elemento X di S copre un elemento Y di S se R(Y,X) e per nessun Z in S
(Z ≠ X) AND (Z ≠ Y) AND R(Y,Z) AND R(Z,X).

DEFINIZIONE 3 - (Peso) Per un ordinamento parziale (S,R) il peso degli oggetti in S è una definizione da S all'insieme dei numeri naturali tale che:

1 - (R(X,Y) AND (X ≠ Y) ⇒ peso(X) < peso(Y)
2 - Se X copre Y allora peso(X) = peso(Y)+1.
Gli oggetti minimali di S hanno peso 1.

LEMMA 1 - Definiamo la relazione
GENERAL(X,Y) = [(X=Y) OR (GENERALIZE(X) = Y)]

La coppia ordinata (O,GENERAL) è un ordinamento parziale.

DEFINIZIONE 4 - (Livello di generalità) Il livello di generalità di oggetto X ∈ O è definito come il peso dell'oggetto nell'ordinamento parziale (O,GENERAL).

4.3.3 Applicazioni agli Ipertesti

Per quanto riguarda i databases, esiste una interessante dicotomia tra lo "schema" del database e le n-uple che costituiscono il database.

Una simile dicotomia esiste negli ipertesti ove la struttura di un ipertesto corrisponde allo schema del database, e l'informazione contenuta nell'ipertesto corrisponde alle n-uple del database.

Per mettere in luce queste caratteristiche, è opportuno classificare le relazioni definite attraverso l'insieme P.

DEFINIZIONE 1 - (Relazioni) Una relazione è:

1 - P(X) ove P ∈ π_1 e P(X) è vero; l'oggetto della relazione è X; oppure
2 - P(X,Y) ove P ∈ π_2 e P(X,Y) è vero; gli oggetti della relazione sono X e Y; oppure

3 - Property(X,Y,Z) ove Property ϵ π_3; l'oggetto della relazione è X.

DEFINIZIONE 2 - (Istanza di una relazione) Le istanze di una relazione sono relazioni i cui oggetti appartengono a I'.

DEFINIZIONE 3 - (Relazioni primitive) Le relazioni primitive sono quelle in cui gli oggetti appartengono a P'.

DEFINIZIONE 4 - (Relazioni ibride) Le relazioni ibride sono relazioni a due argomenti in cui un argomento appartiene a P' e l'altro a I'.

LEMMA 1 - I tre insiemi di relazioni sono mutualmente disgiunti.

Usando questo formalismo, possiamo partizionare un ipertesto in modo da ottenere tre partizioni che riflettano il partizionamento delle relazioni nell'ipertesto. Questo offrirà all'ipertesto caratteristiche simili agli schemi ed insiemi di n-uple dei database relazionali.

LEMMA 2 - Un ipertesto η può essere considerato come l'unione di tre ipertesti:

η_I: Un ipertesto avente $P'[\eta_I] = \varnothing$, $I'[\eta_I] = I'[\eta]$, $A[\eta_I] = A[\eta]$, e $\pi[\eta_I] = \pi[\eta]$
η_P: Un ipertesto avente $I'[\eta_P] = \Delta$, $P'[\eta_P] = P'[\eta]$, $A[\eta_P] = A[\eta]$, e $\pi[\eta_P] = \pi[\eta]$
η_H: Un ipertesto avente $P'[\eta_H] = P'[\eta]$, $I'[\eta_H] = I'[\eta]$, $A[\eta_H] = \varnothing$, e $\pi[\eta_H] = \pi_2[\eta]$

con la condizione che, per un predicato a due argomenti, se gli oggetti appartengono sia a P' sia a I', allora il valore del predicato è falso.

È da notare che η_P corrisponde alla nozione di schema nei database e η_I ad una particolare istanza di schema.

4.3.4 Links

I links offrono un meccanismo di 'navigazione' attraverso la struttura dell'informazione piuttosto che attraverso il contenuto dell'informazione.

I links in questo modello sono definiti da predicati a 2 argomenti tra gli oggetti. I links possono essere usati in un linguaggio basato sul Calcolo dei Predicati per identificare gli oggetti appropriati. Il Prolog è adatto a questo scopo.

5. SISTEMI ESISTENTI

Si possono individuare 4 fondamentali aree di applicazione per i sistemi Hypertext:

1 - *MACRO LITERARY SYSTEMS:* Memex, NLS, Augment, Xanadu, Textnet, Media-Book, Enciclopedia Elettronica.

2 - *PROBLEM EXPLORATION TOOLS:* IBIS, Synview, WE, Outline Processor

3 - *BROWSING SYSTEMS:* ZOG e KMS, INFO, HyperTies, Symbolics Document Examiner

4 - *GENERAL HYPERTEXT TECHNOLOGY:* NoteCards, Intermedia, Boxer, Cref, FileVision, Guide, Hypercard, PlaneText, Thoth-II, HAM, Neptune, Storyspace™.

Vediamo più in dettaglio i singoli sistemi, appartenenti a queste quattro aree di applicazione.

MACRO LITERARY SYSTEMS

Questa area di applicazione riguarda lo studio di tecnologie di supporto per grandi librerie on-line.

Un libro elettronico, che memorizza un grande ammontare di informazione elettronicamente, ha delle capacità molto più vaste di un libro stampato. Innanzitutto può ricercare velocemente l'informazione desiderata, e presentarla con un appropriato mezzo di rappresentazione, in accordo alle esigenze del lettore. Inoltre è possibile cambiare dinamicamente il particolare "punto di vista", ampliando e facilitando, così, la comprensione dell'informazione.

L'intero sistema di pubblicazione in rete costituisce un corpo dinamico, arricchibile dal lettore senza modificare la versione originale del documento.

1. Memex

Vannevar Bush, Presidente della Roosevelt Science Advisor, nell'articolo "As we may Think" (1945), evidenziò la necessità di impegnarsi nel tentativo di meccanizzare i sistemi di letteratura scientifica. In tale articolo egli descrisse una macchina per scrivere appunti e scorrerli in un sistema grafico e di testo on-line. "MEMEX" conteneva una libreria di notevoli dimensioni, aveva molti schermi ed una facility per creare un link etichettato tra due punti nell'intera libreria. Bush descrisse la caratteristica essenziale di Memex, ossia la capacità di collegare due elementi insieme: si rese conto, tuttavia, che le tecnologie esistenti non permettevano l'implementazione di tale sistema.

2. NLS

Engelbart, rifacendosi alle idee espresse da Bush, scrisse nel 1963 "A Conceptual Framework for the Augmentation of Man's Intelletc", in cui propone il sistema H-LAM/T. Nella sua ottica l'utente è un elemento essenziale: utente e computer interagiscono dinamicamente amplificando l'intelligenza naturale umana. Tale visione è proprio comune agli sviluppatori di sistema Hypertext. Nel 1968 queste idee furono implementate come NLS (oN Line Systems) dal centro di ricerca Augmented Human Intellect, come strumento sperimentale per soddisfare certe esigenze di lavoro. I files erano strutturati gerarchicamente in segmenti, chiamati "statements", ognuno dei quali conteneva un identificatore del livello. Si pote-

vano creare un certo numero di links di riferimento tra statements entro e tra files.

NLS enfatizzava tre fondamentali aspetti comuni ai primi sistemi Hypertext:

1 - database di testi non lineari
2 - view filters per selezionare informazioni da questo database
3 - views per strutturare la visualizzazione dell'informazione.

NLS introdusse anche il concetto di editing conferencing distribuito. L'evoluzione di NLS è ora chiamata Augment, ed è un sistema commerciale concentrato su office automation per l'ingegneria del software.

3. **Augment**

Augment, evoluzione di NLS, è stato particolarmente orientato per aiutare i professionisti nel "though knowledge work" (problemi di pianificazione, analisi, disegno in domini complessi). Nel 1970 è stato introdotto il sistema Journal, che permette l'elaborazione del documento da parte di più utenti.

Augment è un sistema di Text processing per un ambiente di rete multiutente; ha un sistema di interfaccia utente che permette una evoluzione facile e flessibile dei linguaggi di comando. I files sono strutturati gerarchicamente e parti qualsiasi di testo sono specificate da indirizzi testuali all'interno di un file qualsiasi.

L'utente può strutturare un file seguendo un proprio modello, componendolo e modificandolo mediante l'uso di comandi (es. move, insert, copy, ...). La struttura gerarchica può essere visitata dall'utente tramite diverse alternative di indirizzamento, tra i quali anche i links, comuni a tutti gli ipertesti.

La possibilità fornita all'utente di muoversi liberamente tra i files permette di raggiungere un particolare "punto di vista" a seconda delle scelte fatte. Il passaggio da un punto di vista all'altro è realizzato tramite un comando di salto ad un link o ad un indirizzo. Augment permette anche con un comando l'attivazione simultanea di più links, costituenti un cammino.

L'utente, durante il lavoro, ha a disposizione una finestra per il documento scelto ed una per lo sviluppo dei suoi commenti e note. Egli può facilmente creare links tra le due finestre e le modifiche verranno rese note a tutti gli utenti interessati.

Augment usa la parola "statement" per riferirsi ad un nodo.

L'interfaccia utente offre infine la possibilità di eseguire procedure.

4. **Xanadu**

Il termine HYPERTEXT venne coniato da Nelson, nel 1960, nella descrizione di Xanadu. Nelson stesso sviluppò le idee di Engelbart, ma concentrò il proprio interesse verso la creazione di un ambiente letterario unificato su scala globale. Soltanto il documento originario ed i cambiamenti fatti ad esso venivano salvati. Il sistema facilmente ricostruiva le precedenti versioni dei documenti. Unità fondamentale era il documento, che poteva avere finestre aperte su altri documenti.

Xanadu è stato implementato su Unix ed è disponibile in diverse forme, incluso come servizio on-line.

5. **Textnet**

Randall Trigg scrisse la prima dissertazione sui sistemi Hypertexts. In questa tesi egli descrive il suo sistema Textnet come supporto ad un testo non lineare, in cui i documenti sono organizzati come "parti primitive di testi collegate da links tipizzati per formare una rete simile per molti aspetti ad una *rete semantica*". Dalla sua tesi:

> "A mio parere, il risultato logico ed inevitabile della rivoluzione del computer sarà il trasferimento al computer di tutte le attività, che coinvolgono la manipolazione di testi, trasformando così la comunicazione all'interno della comunità scientifica. Tutti gli scritti, critiche, riferimenti saranno realizzati on-line. In questo modo gli utilizzatori troveranno riviste specializzate, articoli, technical reports di università distanti, memorizzati su una rete nazionale computerizzata e distribuita. Nuovi scritti verranno inseriti su questa rete spesso con la collaborazione di più autori".

Textnet implementa due tipi base di nodi: "chunks", che contengono testi, e "Tocs", che organizzano gerarchicamente altri nodi.

In questo modo vengono realizzati sia alberi gerarchici, sia grafi non gerarchici (con links tipizzati).

Trigg propose inoltre una specifica tassonomia di tipi di links: un tipo di link per ogni commento. Inoltre Textnet supporta la definizione di liste di nodi ordinate in un cammino, usate per scorrere concatenazioni lineari di testi e stamparle, evitando al lettore la necessità di dover decidere ad ogni passo quale link scegliere (viene dato un cammino per default).

6. **Media Mix Book**

Questo nuovo libro elettronico permette al lettore di reperire l'informazione in modo efficiente, ed aiuta a capire i contenuti grazie ad una opportuna rappresentazione dell'informazione. Sono presenti delle facilities utili a stimolare il processo di pensiero del lettore e per aiutarlo a creare nuove idee.

Le facilities offerte dal sistema per realizzare queste caratteristiche sono: - traverse, - agents, - customization, -multimedie representation.

Gli "agents" sono persone che appaiono nel libro elettro-

nico per spiegare come analizzare l'informazione, come generalizzarla o come specializzarla.

Essi possono spiegare i contenuti da diversi punti di vista, facilitandone la comprensione da parte del lettore. Gli agenti sono caratterizzati in due categorie, in accordo al tipo di interazione con il lettore:

— answer agent; un agente di questo tipo interpreta e spiega i contenuti, è in grado di dare un riassunto dell'informazione o una spiegazione più dettagliata, in accordo alle richieste del lettore;

— co-reader agent; un agente di questo tipo discute i contenuti, ponendo domande o rispondendo a quesiti; il lettore deve soltanto osservare il dialogo.

Grazie agli agenti, il lettore può chiaramente cogliere la distinzione tra fatti e opinioni dell'autore.

Customization è una funzione con la quale può indicare un metodo per compilare l'informazione (ad esempio vi è una differente compilazione dei dati a seconda che il lettore sia inesperto o esperto sull'argomento).

Infine, Multimedia representation permette di rappresentare l'informazione utilizzando dati di natura diversa, come testi, grafi, immagini, suoni e voci. Una rappresentazione audio-visiva è un notevole passo in avanti rispetto a quella offerta da un libro stampato in quanto facilita la comprensione dell'informazione.

Grazie alla facility Traverse il lettore può muoversi anche nello spazio semantico dell'informazione, ottenendo diversi significati cambiando il particolare "Punto di vista" con cui accede all'informazione. Essa offre due tipi di attraversamento dell'informazione:

— inner category browsing: è un metodo di attraversamento gerarchico, in accordo a punti di vista; permette di modificare il particolare punto di vista oppure di passare a descrizioni più dettagliate

— inter category browsing: è un metodo di attraversamento non gerarchico, che permette all'utente di andare direttamente nel punto desiderato selezionando certe parole chiave che si trovano nel testo.

7. Enciclopedia Elettronica

Il pacchetto "Enciclopedia Elettronica" è il risultato finale di un progetto di ricerca nato nel 1985 all'interno del Dipartimento di Scienze dell'Informazione dell'Università Statale di Milano sotto la guida del Prof. G. Degli Antoni.

Scopo della ricerca, realizzata attraverso la tesi di un gruppo di laurendi, era la realizzazione di un sistema di consultazione integrata di testi, immagini, suoni ed animazioni che utilizzasse come supporto di memorizzazione la nuova tecnologia ottica dei CD-ROM (Compact Disc Read Only Memory).

Il pacchetto "Enciclopedia Elettronica" è in realtà composto da tre diverse realtà:

— "Text Mapper" per l'indicizzazione dei testi;

— "Image Link" per l'indicizzazione di immagini, tabelle, disegni;

— "Enciclopedia Elettronica" per la consultazione dei dati memorizzati su CD-ROM ed indicizzati con i pacchetti sopra menzionati.

PROBLEM EXPLORATION SYSTEMS

Questi sistemi, altamente interattivi, forniscono responsi rapidi ad una piccola collezione di comandi specializzati nel trattamento della informazione.

Una caratteristica importante di quasi tutti questi tools è la possibilità di sopprimere dettagli a vari livelli specificati dall'utente.

1. Ibis

Horst Rittel ed i suoi studenti hanno introdotto la nozione di Issue Based Information Systems (IBIS), un metodo per manipolare l'analisi di problemi complessi. La prima versione di IBIS gira su Apple PC, ed ora è stata portata su Sun Workstation. È stata poi sviluppata dalla MCC/Software Technology Program una versione grafica, chiamata gIBIS, su Sun.

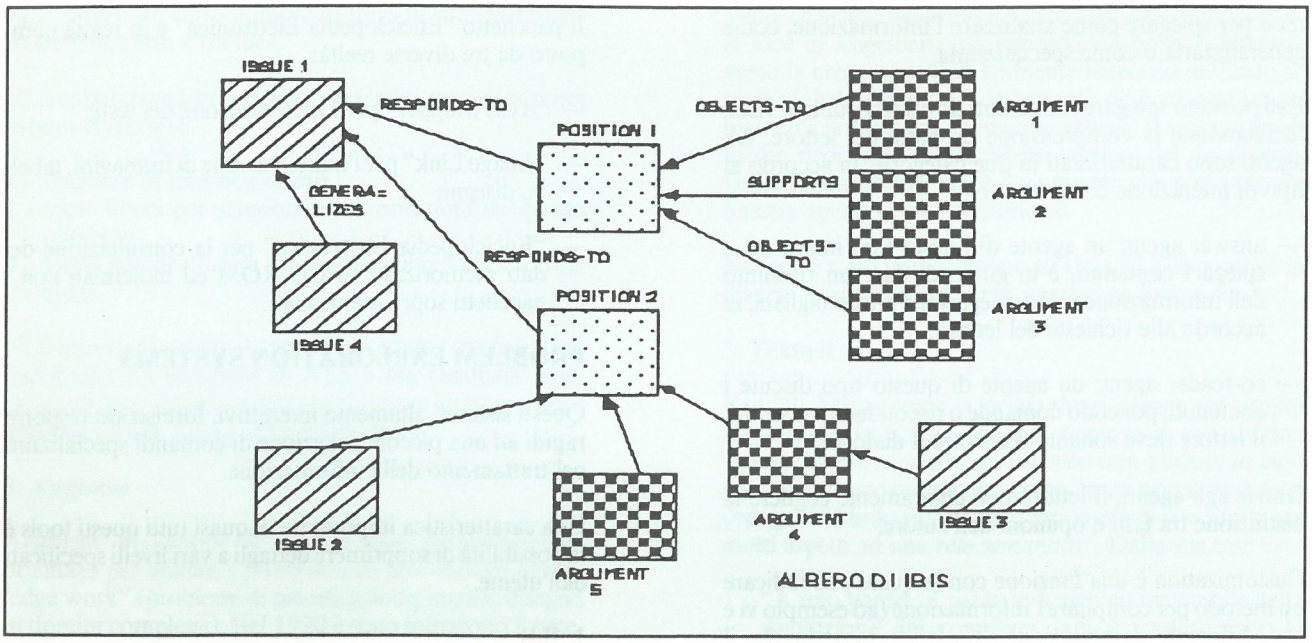
Rittel descrive i cosiddetti "wicked problems" come problemi che non possono essere risolti nel modo tradizionale.

Il solo modo per capire questi problemi è quello di risolverli, le soluzioni non possono essere considerate giuste o sbagliate, queste hanno infatti solamente un grado di sufficienza. Inoltre la soluzione di tali problemi richiede a coloro che ne sono coinvolti di esprimere i propri punti di vista, argomentandoli ed eventualmente modificandoli.

Il vantaggio fondamentale di questo sistema è quello di rendere possibile la comprensione comune dei principali aspetti del problema. IBIS è stato disegnato come supporto a queste conversazioni di design-planning, risultando così un connubbio tra sistemi di teleconferenza diversi ed ipertesti.

Il modello offerto da IBIS si focalizza sulla definizione delle questioni ("issues") chiavi del problema. Ciascuna questione può avere molte posizioni ("positions"), ove una posizione è una asserzione che serve a risolvere una questione. Ciascuna posizione di una questione può avere uno o più argomenti ("arguments") che o la appoggiano o la obbiettano. Ciascuna questione è radice di un albero, eventualmente vuoto, in cui i figli della questione sono le posizioni corrispondenti, ed i figli delle posizioni sono gli argomenti.

IBIS supporta sia una struttura gerarchica sia non gerar-



In IBIS ci sono tre tipi di nodi: -issue, -position, -argument.

chica dei links. Utilizza 9 tipi di relazioni per collegare i nodi. Il corrente insieme di relazioni è: - responds-to, - questions, - support, - objects-to, - specializes, - generalizes, - refers-to, - replaces.

Il browser visualizza la rete in due modi: una vista globale, in cui l'utente può vedere tutta la rete; una vista locale, per visualizzare i dettagli della rete.

2. Synview

Sviluppato da David Lowe, questo sistema è simile ad *IBIS* nei concetti, ma si sviluppa in una direzione diversa.

Esso propone ai partecipanti di valutare la validità e rilevanza dei propri risultati e argomenti prima di esporli.

La stima è fatta da una votazione quantitativa, ed i valori di tale stima sono mostrati su diversi displays, permettendo ai partecipanti di avere in ogni momento presente la traccia delle argomentazioni con votazione maggiore.

3. We

Un gruppo dell'università del North Carolina a Chapel Hill, ha sviluppato un "writing environment" chiamato WE. La ricerca si basa su un modello cognitivo della comunicazione: la lettura è un processo che analizza un testo lineare per comprenderlo, strutturando i concetti gerarchicamente, e memorizzarlo ad esempio in una rete; la scrittura è invece il processo inverso, in cui una rete strutturata di idee è dapprima organizzata in una gerarchia appropriata e successivamente codificata in un testo lineare.

WE è stato sviluppato in modo da rispecchiare i concetti espressi dalla teoria cognitiva del processo di scrittura. Le principale fasi di scrittura di un documento hanno come supporto una modalità di sistema. Le modalità danno 4:

— network mode: viene utilizzata come supporto per la fase iniziale, fondamentalmente esplorativa, dello sviluppo del documento;

— tree mode: aiuta l'utente a costruire la struttura gerarchica del documento;

— editor mode: permette di associare un testo ai nodi della gerarchia (viene utilizzato Smalltalk text editor);

— text mode: permette di dare una rappresentazione lineare al documento, permettendo di vedere la forma in cui il testo sarà stampato.

WE utilizza un database object-oriented, in cui viene memorizzata tutta la informazione strutturale. Gli oggetti sono di 3 tipi: - struttura, - nodo, - link. Le strutture sono tipizzate, ed il tipo indica se la struttura è un grafo, una gerarchia o un cammino.

Il database attualmente è limitato ad un singolo documento, ma verrà poi esteso.

Ogni modalità di sistema (network, tree, editor, text mode) ha associata una finestra sullo schermo.

La modalità editor fornisce l'accesso ad uno standard text editor. Nelle future estensioni di WE, il sistema supporterà anche editor per altri tipi di dati, come grafi, suoni, e video.

La modalità text costituisce il testo in forma lineare, attraversando l'albero dall'alto verso il basso, da sinistra verso destra, inserendo le etichette dei nodi come titoli, seguiti dai blocchi di testo associati ai nodi.

4. Outline Processor

Un Outline Processor è un word processor specializzato nel trattamento di Outline; per questo i suoi principali comandi consistono nel movimento, nella creazione e nella modifica di entrate di outline. Una delle caratteristiche fondamentali è la possibilità di sopprimere i più bassi livelli di dettaglio. Come in *NLS* di Engelbart, l'utente può vedere i livelli più alti dell'outline, oppure scorrere l'albero, aprendo solo quelle entrate, che sono rilevanti per l'idea sulla quale egli sta lavorando. Inoltre, aprendo ogni entrata può avere un corpo di testo di qualsiasi lunghezza associato, e l'utente ha la possibilità di visualizzare o meno questo corpo.

Il primo Outline Processor ad essere stato realizzato fu il Thinktank (1984), esteso poi sotto altri nomi (Max Think, Executive Writer/Executive Filer, Thor, Framework, Kamas, Fact, Cruncher, Freestyle, Idea!, PC-Outline, e More).

Nessuno degli Outline Processor fornisce links ad icona con il mouse, ma hanno in comune con gli Hypertexts la proprietà di trattare segmenti di testi come principale classe di oggetti, e di manipolare idee gerarchicamente.

BROWSING SYSTEMS

I sistemi di questa area sono stati creati per applicazioni coinvolgenti grandi quantità di informazioni e richiedenti un facile accesso a tali informazioni. I problemi dominanti nello sviluppo di questi sistemi sono la facilità di uso e di apprendimento.

1. ZOG e KMS

Lo sviluppo di ZOG è iniziato nel 1972, alla Carnegie-Mellon University. È costituito da un grande database di piccoli segmenti (grandezza schermo), visualizzati uno per volta. Zog è stato sviluppato con l'obiettivo di servire una grande comunità di utenti, e quindi è stato creato per funzionare sui terminali di un sistema time sharing. Nel 1981 Donald McCracken e Robert Akscyn hanno sviluppato un successore commerciale di Zog, chiamato KMS (Knowledge Management System), per servire reti di workstation eterogenee.

In KMS un database consiste di un insieme di "frames", tra loro collegati, contenti testi, grafi ed immagini. Elementi di testo possono anche attivare programmi.

Ogni frame ha un unico nome, che consiste di due parti: una parte formata da caratteri alfabetici, ed una parte da caratteri numerici. La parte alfabetica è il nome del "frameset" al quale il frame appartiene.

Sull'angolo sinistro superiore del frame è visualizzato il titolo, che offre una breve descrizione del contenuto. Sotto il titolo si trova il corpo del frame.

Le linee successive sono elementi testuali chiamati "tree items", che sono collegati a frames a livello inferiore. La presenza di un link è indicata da un piccolo cerchio alla sinistra dell'elemento.

Sul lato inferiore destro ci sono gli "special items" che iniziano col carattere "©", e sono utilizzati per note, commenti, comandi di formattazione.

Infine l'ultima riga della finestra contiene i "command items", usati per richiamare programmi.

KMS ha un solo tipo do nodo, il frame è di ampiezza schermo.

La sorgente di un link è uno specifico elemento in un frame (tree items). I links non sono inseriti entro il testo come nei tradizionali ipertesti, bensì vi è una scissione tra contenuti di un frame e links ad altri frames.

La destinazione di un link in KMS è un intero frame (non è presente il concetto di "regione" e "punto").

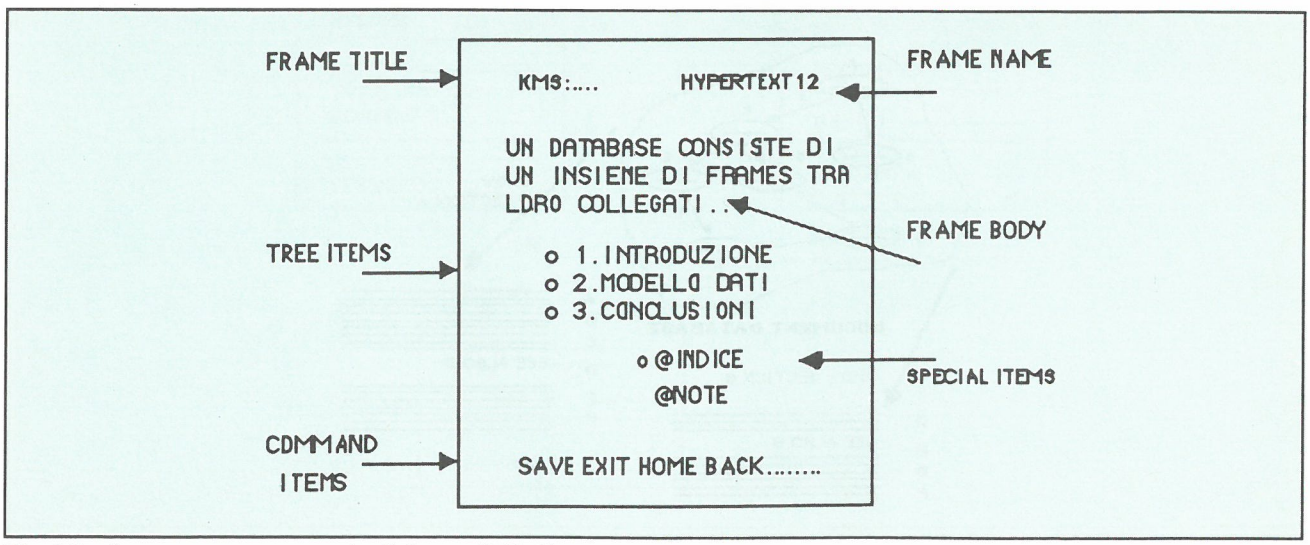


Fig. 1 La figura mostra un esempio di frame di KMS.

Esistono due tipi di links:

— tree items, per il collegamento con frames a livello inferiore in una gerarchia

— special items, per il collegamento con materiale subordinato, come commenti e cross references.

La struttura è generalmente gerarchica, con la possibilità di riferimenti incrociati.

In KMS l'interfaccia utente è stata realizzata senza utilizzare finestre sovrapposte, visualizzando un solo frame alla volta, e cercando di dare un ambiente con poche distinzioni concettuali. Non esiste infatti differenza tra modalità di editing e di navigating; l'utente può modificare il frame in qualsiasi momento, i cambiamenti vengono salvati automaticamente.

Il frame selezionato con un link appare sullo schermo al posto di quello precedente. la possibilità di visualizzare soltanto un frame alla volta crea disorientamento nell'utente. In KMS sono stati ridotti i tempi di attraversamento in modo da velocizzare il passaggio da un frame all'altro.

2. **Emacs Info Subsystem**

Info, molto simile a *Zog*, è un supporto per il text editor Emacs. Ha un più semplice insieme di comandi standard ed il controllo dell'input avviene su singole lettere o brevi comandi digitati alla tastiera.

Principalmente gerarchico può passare da un punto all'altro della gerarchia digitando il nome del destinatario. Vi è il problema del disorientamento dell'utente, essendo visualizzato un solo frame alla volta.

3. **Hyperties**

Il progetto HyperTIES della Maryland University è stato sviluppato dal 1983 in due direzioni: come tool pratico e semplice per scorrere un database per l'istruzione, e come piattaforma sperimentale per studi sul disegno di interface per ipertesti. Inizialmente il progetto si chiamava TIES (The Interactive Enciclopedia System), il nuovo nome fu scelto per evidenziare le relazioni del sistema con i concetti degli ipertesti.

Hyperties consiste di: - un browser; - un Tool per comporre nuovi articoli; - un database. Hyperties ha come scopo principale quello di aiutare autori ad organizzare idee ed a migliorare il loro stile. L'utente può leggere o interrogare il database sull'argomento selezionato.

In Hyperties unità fondamentale sono brevi articoli (50/100 parole), collegati da un qualsiasi numero di links. Hyperties guida l'autore a scrivere un titolo ed un corpo. Inoltre l'articolo ha una breve descrizione (5/25 parole) che può essere visualizzata, evitando così all'utente di richiamare l'intero articolo per sapere di cosa esso tratti. L'autore può segnare dei punti nel testo come "references", circondandoli con delle "tilde".

I links sono parole o frasi evidenziate (ad esempio in reverse) nell'articolo.

Il sistema HyperTies tiene traccia del cammino dell'utente sulla rete, rendendo possibile un facile ritorno dal cammino esplorato.

4. **Symbolics Document Examiner**

DocumentExaminer è parte della Symbolics Genera, un ambiente di sviluppo di software per Symbolics computers (1985).

Il progetto di Document Examiner è stato sviluppato con lo scopo di rendere semplice e funzionale la ricerca ed uso di informazioni in un insieme molto grande di documenti.

La documentazione è organizzata come un database di moduli. Lo scrittore determina la natura della modularità, che dipende dal particolare soggetto. Il modulo può avere qualsiasi dimensione e può contenere oggetti di ogni tipo. tali caratteristiche vengono scelte dallo scrittore, in accordo alle eventuali esigenze del lettore.

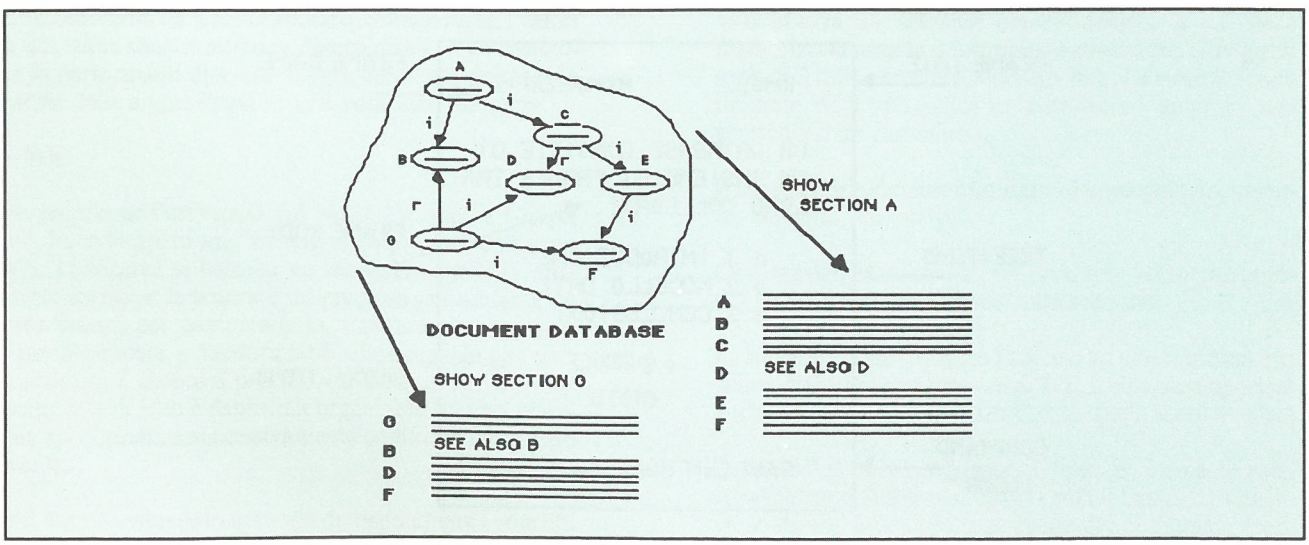


Fig. 2

La figura mostra il cammino nel database. I nomi dei nodi sono espressi con le lettere A ... G. I links tra i nodi sono rappresentati dalle frecce, e sono etichettati con "i" per indicare inclusione e con "r" per indicare un cross reference.

In Document Examiner i links sono direzionali, collegano un intero record, che rappresenta internamente il modulo, o un punto in esso ad un altro record. Un record può "chiamare" qualsiasi altro record attraverso un link.

Document Examiner è una utility basata sull'uso di finestre, integrata con il resto dell'ambiente software Symbolics.

La caratteristica più importante dell'interfaccia è quella di presentare al lettore il materiale come un libro, permettendo una notevole semplicità di uso.

Una finestra è divisa in pannelli, con differenti utilizzazioni:

— Viewer: la maggior parte dello schermo è usata per mostrare l'argomento, dando al lettore la sensazione di un libro;

— Command pane: l'area inferiore dello schermo contiene un menu fisso di comandi;

— Candidates pane: "candidates" è il termine usato per indicare un insieme di nomi di records che rispondono ad una richiesta dell'utente;

— Bookmarks pane: questa area dello schermo mantiene la cronologia dei records letti;

— Overview window: è una finestra temporanea che visualizza il contesto dell'argomento, mostrando i links di inclusione e tutti i links uscenti.

GENERAL HYPERTEXT TECHNOLOGY

I sistemi di quest'area si riferiscono ad una o più applicazioni, ma il loro obiettivo principale è quello di fornire un mezzo di sperimentazione con gli stessi ipertesti visti come tecnologia.

1. HAM

La Hypertext Abstract Machine è un server general-purpose, transactio-based per un sistema Hypertext in un ambiente a rete. HAM è basata sulla macchina astratta usata da NEPTUNE. Delisle e M. Schwartz nel sistema *Neptune*, sviluppato presso il laboratorio di ricerca Tektronix. Essendo un mezzo di memorizzazione a basso livello, HAM offre un modello generale e flessibile che può essere usato in molte e differenti applicazioni di ipertesti.

HAM utilizza 5 differenti tipi di oggetti (grafi, contesti, nodi, links e attributi) organizzati gerarchicamente, mantiene una storia di questi oggetti e permette un accesso selettivo attraverso meccanismi di filtro.

Un "grafo" è l'oggetto HAM a più alto livello, contiene contesti, nodi, links, attributi.

Generalmente un grafo contiene tutta l'informazione su un argomento generale.

I "contesti" partizionano i dati entro un grafo. Ciascun contesto ha un contesto genitore e zero o più contesti figli. Quando viene creato un grafo una radice contesto inizia l'albero.

Un contesto non dipende dall'informazione contenuta nel contesto genitore e può contenere zero o più nodi e links.

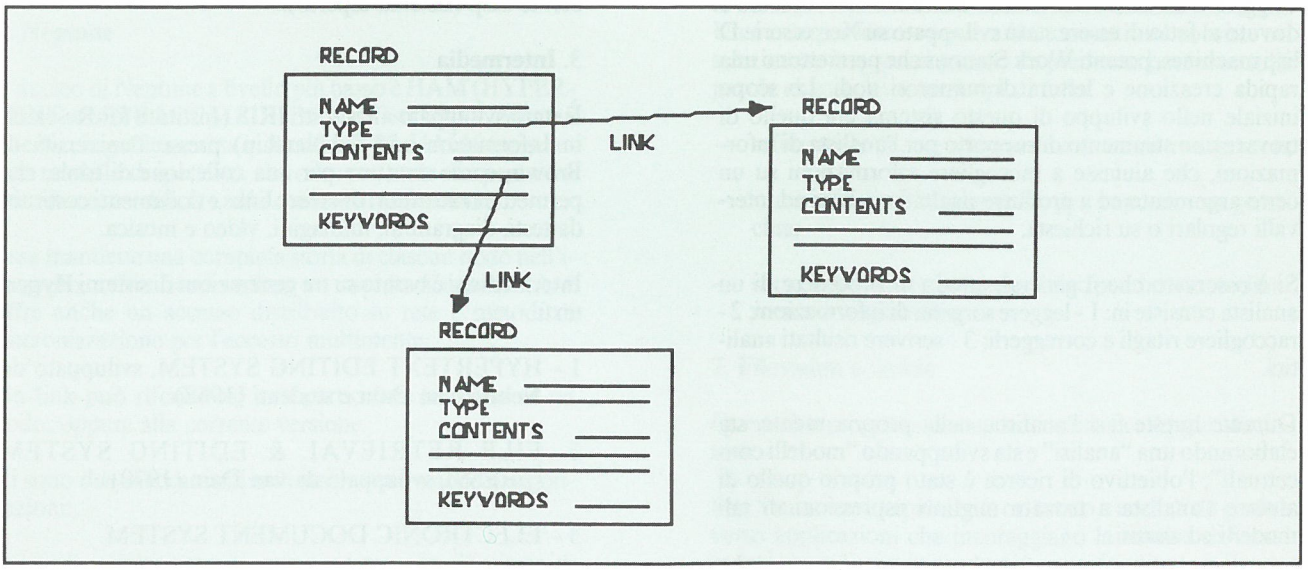


Fig. 3 La figura mostra un link da un punto in un record ad un altro record.

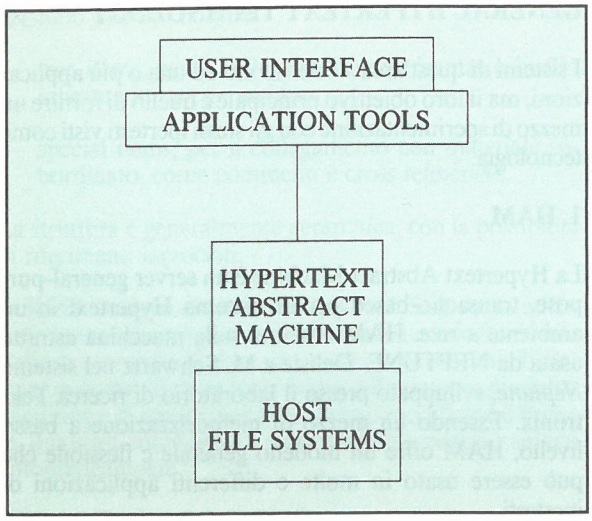


Fig. 4 Architettura generale di un sistema Hypertext.

Gli "attributi" possono essere riferiti a contesti, nodi, links. Le coppie attributo/valore danno una semantica agli oggetti di HAM, permettendo di rappresentare proprietà degli oggetti a seconda della applicazione. Gli attributi sono anche usati nei predicati dei filtri. Tali filtri recuperano in modo selettivo l'informazione da un grafo. Un nodo contiene dati arbitrari. I nodi sono collegati da links.

Un link può essere seguito in entrambe le direzioni. Un link "cross context" mette in relazione due nodi in differenti contesti. Links di questo tipo sono utili per condividere dati tra due contesti. Un link in HAM è un oggetto molto generico, permette così di definire tipi e schemi di collegamento a seconda delle applicazioni.

2. Note Cards

Note Cards è stato sviluppato alla Xerox PARK da R. Trigg, T. Moran e F.G. Halasz. Parte del suo successo è dovuto al fatto di essere stato sviluppato su Xerox serie D lisp machines, potenti Work Stations che permettono una rapida creazione e lettura di numerosi nodi. Lo scopo iniziale nello sviluppo di questo sistema era quello di trovare uno strumento di supporto per l'analista di informazioni, che aiutasse a raccogliere informazioni su un certo argomento ed a produrre risultati analitici ad intervalli regolari o su richiesta.

Si è osservato che il generale modo di procedere di un analista consiste in: 1 - leggere sorgenti di informazioni; 2 - raccogliere ritagli e correggerli; 3 - scrivere risultati analitici.

Durante queste fasi l'analista, nella propria mente, sta elaborando una "analisi" e sta sviluppando "modelli concettuali"; l'obiettivo di ricerca è stato proprio quello di aiutare l'analista a trovare migliori espressioni di tali modelli ed analisi.

Il sistema è basato su due costrutti principali: notecards e links. Una notecard ha un nome.

Una finestra particolare è quella di editing, ove la notecard visualizzata può essere modificata dall'utente.

Esistono diversi tipi di notecards, differenziate dalla natura del contenuto; inoltre il sistema offre una facility per creare nuovi tipi di cards. Un tipo di notecard è il "browser", che contiene un diagramma strutturato di cartelle. I titoli dei nodi della rete sono campi sensibili al mouse, che permettono con la loro attivazione l'attraversamento della rete.

"Fileboxes" sono cartelle specializzate per organizzare e catalogare notecards e fileboxes stesse, introducendo un nuovo livello di gerarchia.

In NoteCard i links connettono notecards individuali in una rete. Ogni link è una connessione, avente un certo tipo, diretta da una cartella sorgente ad una cartella destinazione. Il tipo del link è una etichetta, scelta dall'utente, che specifica la natura della relazione delle due cartelle. I links sono ancorati ad una locazione particolare della sorgente tramite una icona, ed hanno come destinazione una cartella intera. In NoteCards non esiste il concetto di "punto" o "regione" all'interno di una cartella destinazione.

L'attraversamento della rete di notecards è reso possibile dalla speciale notecard chiamata "browser", ove è contenuta la struttura. Una volta visualizzata questa cartella, è possibile seguire i links semplicemente posizionandosi sul nodo desiderato. Questo metodo di "naviting" risulta però difficoltoso e poco chiaro se la rete è costituita da molti nodi, in quanto l'utente può perdersi facilmente, non raggiungendo così l'informazione desiderata.

L'interfaccia utente include un'interfaccia per programmatore con oltre 100 funzioni Lisp per permettere la creazione di nuovi tipi di cartelle per sviluppare programmi Lisp e per integrare NoteCards in un altro ambiente Lisp (sistema esperto).

3. Intermedia

È stato sviluppato all'istituto IRIS (Institute for Research in Information and Scholarship) presso l'università di Brown, come struttura per una collezione di tools, che permettano ad autori di creare links e documenti costituiti da testi, diagrammi, immagini, video e musica.

Intermedia si è basato su tre generazioni di sistemi Hypertext:

1 - HYPERTEXT EDITING SYSTEM, sviluppato da Nelson, van Dam e studenti (1968)

2 - FILE RETRIEVAL & EDITING SYSTEM (FRESS), sviluppato da van Dam (1970)

3 - ELECTRONIC DOCUMENT SYSTEM.

Le applicazioni includono Itertext, un Text Processor;

Interdraw, un editor grafico; Interval, un timeline editor, che permette agli utenti di organizzare le informazioni in sequenza temporale; Interspec, un visore di sezioni di oggetti in 3D; Interpix, per scandire immagini. Intermedia si occupa soprattutto di aiutare l'utente a gestire efficientemente l'ambiente Hypertext.

Ad esempio, dato che molti links uscenti dallo stesso punto di un documento possono confondere il lettore, Intermedia visualizza una singola icona, la quale, per mezzo del mouse, può mostrare gli specifici links, i loro nomi, ed i nodi-destinazione. Intermedia propone anche un costrutto chiamato "webs" (reti), per gestire le dipendenze dal contesto; ogni link appartiene ad una o più reti ed è visibile soltanto quando una di tali reti è attiva.

Un altro aspetto importante è il modo in cui è possibile scorrere una rete che può includere migliaia di nodi.

Ci sono due tipi di displays:

— una mappa globale, che mostra l'intero iperdocumento e ne permette l'attraversamento;

— una mappa locale, che visualizza un singolo documento, mostrando i suoi links e i nodi più vicini alla rete.

Inoltre nodi e links possono essere visualizzati secondo molti livelli di dettaglio.

Intermedia fornisce un supporto elettronico di processi cooperanti di conoscenza, permettendo agli autori di creare, annotare, e linkare informazioni da vari media, come testi, grafici, suoni, time line, fogli elettronici etc.. Mentre i sistemi tradizionali permettono un accesso sequenziale, i sistemi Intermedia permettono agli autori di aggiungere commenti allo stesso corpo nello stesso momento, visualizzando un mondo complesso in dinamica evoluzione.

4. **Neptune**

Il nucleo di Neptune a livello più basso è HAM (HYPERTEXT ABSTRACT MACHINE) in C su UNIX, con una interfaccia grafica scritta in Smalltalk-80. HAM è un generico modello di ipertesto, che offre una serie di operazioni per creare, modificare, ed accedere a nodi e links.

Essa mantiene una completa storia di ciascun nodo nell'ipergrafo, offrendone inoltre un rapido accesso. HAM offre anche un accesso distribuito su rete e metodi di sincronizzazione per l'accesso multiutente.

Un link può riferirsi ad una particolare versione di un nodo, oppure alla corrente versione.

Vi sono due meccanismi utili per la costruzione di applicazioni:

— nodi e links possono avere un numero illimitato di coppie attributo-valore;

— speciali predicati sono inclusi per richiedere i valori di queste coppie, permettendo che livelli di applicazione più alti definiscano propri meccanismi nel grafo.

L'interfaccia offre molti scorritori, tra cui il "Graph Browser", che permette la visualizzazione di un sottografo; il "Document Browser" per l'attraversamento di strutture gerarchiche; il "Node Browser" per accedere ad uno specifico nodo nel documento.

5. **Boxer**

Boxer è un linguaggio di programmazione altamente interattivo, di uso molto semplice, scritto da di Sessa.

Esso utilizza la nozione di "box" come unità fondamentale di informazione nel sistema, che può connettere altri boxes o dati anche grafici. Boxer offre un tipo di facility cross-reference, chiamata "port", che consiste nella visualizzazione del box a cui ci si riferisce nel punto in cui c'è il riferimento. Un box può essere cambiato attraverso uno dei suoi ports e tutti i ports riflettono tale cambiamento. Le gerarchie sono espresse in modo naturale tramite boxes: le finestre di nodi a livello più basso sono innestate direttamente entro i genitori.

6. **Cref**

CREF (Cross Referenced Editing Facility) è un prototipo, creato da Pitman, di text e graphic editor, sviluppato come tool per analizzare risultati di esperimenti psicologici (protocolli), ed usato anche come modello di Hypertext.

I nodi del sistema sono frammenti di testo, detti "segmenti".

Tali segmenti sono organizzati in serie lineari, possono avere parole chiave e molti tipi di links ad altri segmenti.

Cref organizza i segmenti in "collezioni", che possono essere definite implicitamente da un predicato (collezione astratta), o esplicitamente da una lista (collezione statica). Un modo per formare una collezione astratta consiste nel selezionare i segmenti usando un predicato booleano sulle parole chiave. Vi sono 4 tipi di links:

— reference links, per riferimenti incrociati tra segmenti;

— summarizes references links, per imporre una gerarchia;

— supersedes links e precedes links, per dare un ordinamento ai segmenti.

7. **Filevision e Guide**

Questi due programmi sono stati scritti per Apple Macintosh.

File Vision è principalmente orientato a nodi di un grafo e verso applicazioni che incoraggiano la tecnica di visual indexing, vediamo in modo più approfondito le sviluppo di GUIDE.

Lo sviluppo di Guide iniziò nel 1982, presso l'università del Kent a Canterbury, come progetto di ricerca con lo scopo di trovare une efficiente rappresentazione di documenti tramite computer. Guide non iniziò come progetto di ipertesto. La prima implementazione è stata sviluppata su UNIX (1983). Nel 1984 OWL si interessò al progetto, e decise di implementarlo come prodotto commerciale su Apple Macintosh. Guide è ora disponibile anche su IBM PC. Guide non offre le capacità grafiche di file vision, ma supporta efficientemente una organizzazione ad ipertesto di dati testuali.

Possiamo considerare un intero documento come un nodo. Le relazioni tra parti di testo sono rappresentate tramite "bottoni", links che hanno come destinazione l'intero nodo contenente l'informazione associata al bottone.

Un bottone è una speciale area dello schermo selezionabile per mezzo del mouse.

Vi sono tre tipi di links:

— replacement links, per sostituire il testo nella corrente finestra con il testo a cui punta il link;

— note links, per visualizzare il testo della destinazione in una apposita finestra;

— references links, per aprire una nuova finestra con il testo destinazione.

È possibile raggruppare bottoni di tipo replacement tramite il bottone inquiry. Se un bottone inquiry viene selezionato, l'intero inquiry è sostituito con il replacelent corrispondente. Questo permette di avere, come sorgente del link, una regione piuttosto che un punto, e fornisce all'autore un meccanismo di presentazione dell'informazione secondo diversi punti di vista. In particolare, se l'inquiry copre tutto lo schermo, è possibile simulare un sistema basato su "frames".

In Guide il più importante meccanismo di attraversamento di un insieme di documenti è il replacement button. Il testo che costituisce un bottone può a sua volta contenere altri bottoni. Il lettore esplora un documento espandendo successivamente i bottoni, fino a quando raggiunge il livello di dettaglio desiderato. È possibile tornare indietro, a livelli di dettaglio inferiori. Un altro meccanismo di linking, tipico degli ipertesti, è quello che utilizza il reference button. Tale bottone, quando viene selezionato, provoca un salto in un altro punto del documento in esame, oppure, se riferito ad un altro documento, sovrappone una nuova finestra contenente la destinazione. Questo meccanismo crea però delle difficoltà all'utente, in quanto non risulta immediato e naturale il metodo per ripristinare le finestre precedenti, nel caso in cui ci siano più finestre aperte.

L'interfaccia è molto semplice e permette di inserire nel testo grafi e disegni, utilizzando specifici tools.

8. **Hypercard**

Hypercard è uno strumento software che permette di organizzare informazioni in modo associativo, utilizzando la tipica interfaccia Macintosh. Esso permette all'utente una consultazione veloce ed intuitiva di una grande quantità di informazione, offrendogli inoltre la possibilità di creare una visione personale dell'informazione.

Hypercard è strutturato ad oggetti. Ogni oggetto è una struttura informativa "intelligente"; può avere cioè un programma associato che lo specializza funzionalmente. Gli oggetti base sono: la scheda, lo stack, il bottone.

Hypercard è per alcuni aspetti simile a *NoteCards*. Esso fornisce uno speciale supporto per links "eseguibili", dandogli la caratteristica di linguaggio di programmazione. Il linguaggio di Hypercard è Hypertalk.

La "scheda" è l'unità informativa di Hypercard, e può contenere informazioni eterogenee: testo, grafica, immagini digitalizzate, suoni. Ogni schermata su Macintosh è una scheda.

Lo stack è una collezione consistente di schede, e rappresenta un documento.

Un link è rappresentato in Hypercard tramite l'oggetto "bottone", che permette di collegare parti di informazioni di una scheda. Il bottone definisce un'area dello schermo, sensibile al mouse, con specifiche funzionalità.

Nel menu, visualizzato nella prima linea dello schermo, è presente la funzione GO, contenente i seguenti comandi per gestire l'attraversamento di schede:

— Back, per tornare alla scheda precedente

— Home, per tornare alla prima scheda dell'Home stack

— First, Prev, Next, Last

— Recent, che memorizza le 42 più recenti cartelle.

Hypercard offre all'utente una interfaccia molto semplice, e permette l'accesso a tutte le risorse di elaborazione di Macintosh, senza alcuna necessità di conoscenza dei livelli più bassi della macchina (hardware, sistema operativo, toolbox).

9. **Plane Text**

È stato sviluppato nell MCC Software Technology Program.

Si basa su file System UNIX e su Sun Sunview Window Manager.

Ciascun nodo è un file, i links appaiono come nomi tra parentesi graffe, e possono essere visualizzati o meno. I links sono implementati come puntatori salvati su file separati.

Questo sistema permette così l'integrazione di Hypertext nell'ambiente computazionale UNIX, includendo tools come Mail e News. Inoltre il sistema di directory UNIX fornisce un utile meccanismo per costruire gerarchie.

10. **Thoth-II**

George H. Collier ha sviluppato Thoth-II con lo scopo principale di introdurre il concetto di semantica nei sistemi Hypertext, utilizzando grafi diretti etichettati. Egli vede infatti i nodi come rappresentanti di oggetti, i links come rappresentanti relazioni tra questi oggetti. In questo modo Collier vuole rendere esplicita l'informazione contenuta implicitamente negli ipertesti.

Thoth-II si basa sul concetto di ipertesto visto come un grafo diretto. Il sistema tratta gli ipertesti come oggetti immersi in un grafo diretto ed etichettato. Tale grafo è una rete semantica, utilizzata per rappresentare il dominio di informazione del testo.

Una interpretazione semantica del testo è ottenuta confrontando frasi e connettendo a queste nodi nel grafo.

La semantica della rete offre una guida per l'attraversamento del testo. La struttura non è gerarchica e questo causa un facile smarrimento dell'utente nel grafo.

Thoth-II ha un lessico memorizzato di frasi automaticamente e dinamicamente confrontate con il testo. Ogni frase è collegata con un nodo della rete.

In Thoth-II, diversamente da altri ipertesti, i nodi hanno una esistenza separata dal testo. È possibile creare una rete che non ha parti di testo ad essa associate; questo perchè una rete non è soltanto un metodo per connettere parti di testo, bensì è utilizzata per rappresentare i concetti chiave del documento che lo scrittore sta sviluppando.

I nodi rappresentano una ancora concettuale, intorno alla quale è possibile collezionare parti di testo o connessioni ad altri oggetti. Queste ancore pongono degli agganci nel testo tramite un processo con il quale la rete viene connessa al testo tramite insieme di frasi chiave, rendendo Thoth-II un sistema simile al natural-language processing.

In Thoth-II i links connettono i nodi della rete e il testo è connesso a questi nodi da altri tipi di links.

Il sistema offre tre tipi di links:

— value links, per collegare un nodo con un altro

— text links, per connettere un testo con il rispettivo nodo

— lexical links, per connettere regioni di un testo con un nodo.

Un documento è costituito da questi tre tipi di links e dai rispettivi nodi (fig. 5).

È possibile definire in Thoth-II un altro tipo di link, chiamato backlink, che permette di muoversi nella direzione opposta a quella definita dal link attivato.

L'interfaccia ha due modalità di sistema: browsing mode e text display mode.

I nodi sono visualizzati in due modi, a seconda che essi siano espansi o meno.

L'attraversamento del grafo è realizzato tramite l'espansione dei nodi.

In una finestra corrispondente alla modalità text l'utente può leggere il testo associato ad un nodo.

11. **Storyspace™**

Jay David Bolter e Michael Joice hanno creato in collaborazione con il professor John B. Smith un sistema per la

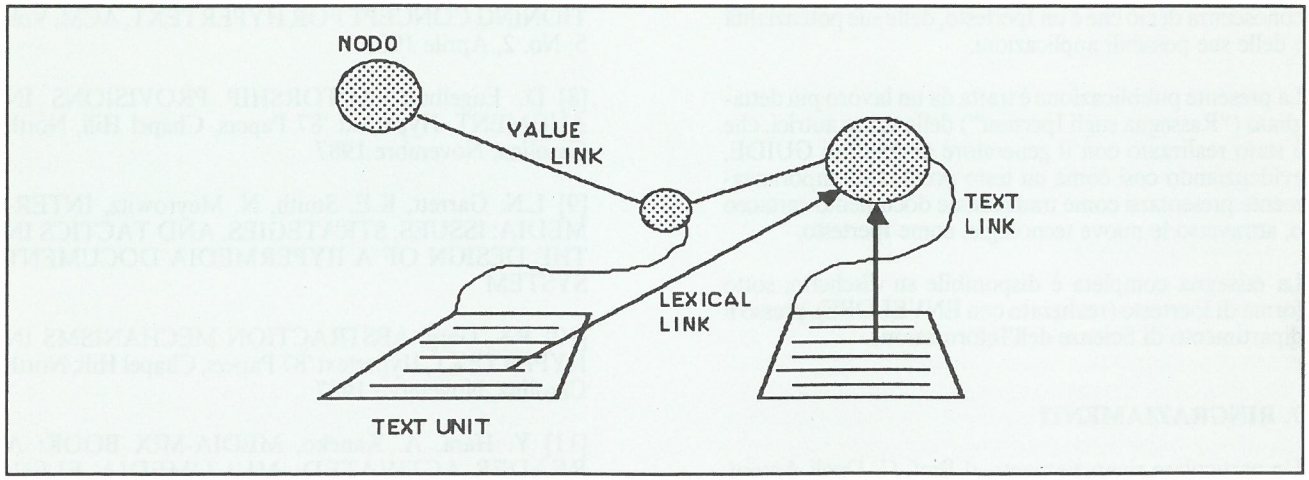


Fig. 5 La figura mostra un diagramma di tipi di strutture, che costituiscono un documento di Thoth-II.

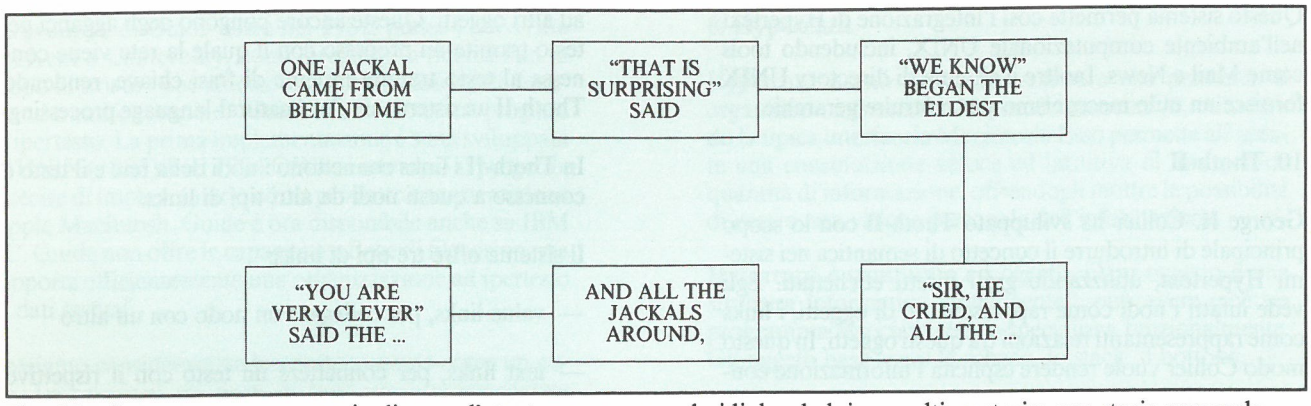


Fig. 6 La figura mostra un esempio di come l'utente possa seguendo i links, da lui prescelti, costruire una storia personale.

prosa narrativa interattiva, chiamato Storyspace™, che implementa lo schema di episodi collegati tra loro tramite links.

Questo sistema di prosa permette al lettore di crearsi la propria novella (fig. 6).

Storyspace ha due modalità: una per l'autore, una per il lettore. L'autore crea la propria novella come una serie di episodi testuali collegati tra loro da links. L'autore ha visualizzata la struttura a grafo dell'ipertesto che sta creando, mentre il lettore vede solo il testo dell'episodio che sta leggendo.

In Storyspace un nodo corrisponde ad un episodio della novella.

Il lettore deve rispondere ad una domanda posta alla fine del testo, questo gli dà la possibilità di passare ad un altro episodio tramite un link, corrispondente alla risposta data. Storyspace è implementato sa Macintosh, offre all'utente un'interfaccia semplice e grafica, con l'uso del mouse il lettore può scorrere il grafo dell'ipertesto per generare la propria storia.

6. CONCLUSIONI

Si auspica, con questo articolo, di aver portato il lettore a conoscenza di ciò che è un Ipertesto, delle sue potenzialità e delle sue possibili applicazioni.

La presente pubblicazione è tratta da un lavoro più dettagliato ("Rassegna sugli Ipertesti") delle stesse autrici, che è stato realizzato con il generatore di Ipertesti GUIDE, evidenziando così come un testo possa contemporaneamente presentarsi come tradizionale documento cartaceo o, attraverso le nuove tecnologie, come Ipertesto.

La rassegna completa è disponibile su dischetto, sotto forma di Ipertesto (realizzato con ENVELOPE), presso il dipartimento di Scienze dell'Informazione.

7. RINGRAZIAMENTI

Un particolare ringraziamento al Prof. G. Degli Antoni, che ha suggerito e coordinato questo lavoro.

8. BIBLIOGRAFIA

[1] R. Akscyn, D. MacCracken, E. Yoder, KMS: A DISTRIBUTED SYSTEMS FOR MANAGING KNOWLEDGE IN ORGANIZATION, Hypertext '87 Papers, Chape Hill, North Carolina, Novembre 1987

[2] P.J. Brown, TURNING IDEAS INTO PRODUCTS: THE GUIDE SYSTEM, Hypertext '87 Papers, Chapel Hill, North Carolina, Novembre 1987

[3] B. Campbell, J.M. Goodman, HAM: A GENERAL PURPOSE HYPERTEXT ABSTRACT MACHINE, Hypertext '87, Chapel Hill, North Carolina, Novembre 1987

[4] J. Conklin, HYPERTEXT: A SURVEY AND INTRODUCTION, I.E.E. Computer, Vol. 20, No. 9, Settembre 1987

[5] J. Conklin, gIBIS: A HYPERTEXT TOOL FOR TEAM DESIGN, Hypertext '87 Papers, Chapel Hill, North Carolina, Novembre 1987

[6] G.H. Collier, THOTH-II: HYPERTEXT WITH EXPLICIT SEMANTICS, Hypertext '87 Papers, Chapel Hill, North Carolina, Novembre 1987

[7] N.M. Delisle, M.D. Schwartz, CONTEXT-A PARTIONING CONCEPT FOR HYPERTEXT, ACM, Vol. 5, No. 2, Aprile 1987

[8] D. Engelbart, AUTORSHIP PROVISIONS IN AUGMENT, Hypertext '87 Papers, Chapel Hill, North Carolina, Novembre 1987

[9] L.N. Garrett, K.E. Smith, N. Meyrowitz, INTERMEDIA: ISSUES, STRATEGIES, AND TACTICS IN THE DESIGN OF A HYPERMEDIA DOCUMENT SYSTEM

[10] P.K. Garg, ABSTRACTION MECHANISMS IN HYPERTEXT, Hypertext '87 Papers, Chapel Hill, North Carolina, Novembre 1987

[11] Y. Hara, A. Kaneko, MEDIA-MIX BOOK: A READER-ACTIVATED, MULTIMEDIA ELECTRONIC BOOK, Agosto 1987

[12] C. Neuwirth, D. Kaufer. R. Chimera, T. Gillespie, THE NOTE PROGRAM: A HYPERTEXT APPLICATION FOR WRITING FROM SOURCE TEXTS, Hypertext '87 Papers, Chapel Hill, North Carolina, Novembre 1987

[13] T. Oren, THE ARCHITECTURE OF STATIC HYPERTEXTS, Hypertext '87 Papers, Chapel Hill, North Carolina, Novembre 1987

[14] B. Schneiderman, USER INTERFACE DESIGN FOR THE HYPERTIES ELECTRONIC ENCYCLOPEDIA, Hypertext '87 Papers, Chepal Hill, North Carolina, Novembre 1987

[15] J.B. Smith, S.F. Weiss. G.J. Ferguson, A HYPERTEXT WRITING ENVIRONMENT AND ITS COGNITIVE BASIS, Hypertext '87 Papers, Chapel Hill, North Carolina, Novembre 1987

[16] F. Vagliasindi, INTEGRAZIONE TESTO-IMMAGINI: UN'APPLICAZIONE NELL'AMBIENTE DELL'ENCICLOPIA ELETTRONICA, Tesi di Laurea presso il Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, anno accademico 1986-87

[17] J. Walker, DOCUMENT EXAMINER: DELIVERY INTERFACE FOR HYPERTEXT DOCUMENTS, Hypertext '87 Papers, Chapel Hill, North Carolina, Novembre 1987

UN ESPERIMENTO CONNESSIONISTA SULLA COMPRENSIONE DEL LINGUAGGIO NATURALE

Stefano Nolfi (+), Domenico Parisi (°)

(+) Fondazione Sigma Tau

(°) Istituto di Psicologia del CNR - Roma

RIASSUNTO

Il presente articolo descrive un approccio al trattamento del linguaggio naturale basato sull'uso di reti neoronali. In particolare si è affrontato il problema dell'apprendimento e dell'assemblaggio corretto delle parole di una fase d'ingresso, generando così una rappresentazione complessiva del significato della frase.
La rete neuronale usata è basata sul modello della 'back propagation', dove l'apprendimento avviene mediante stabilizzazione dei pesi tra le connessioni tra le sue unità, che rappresentano pertanto la conoscenza che il sistema utilizza pre giungere agli assemblaggi corretti.

ABSTRACT

This article describes an approach to natural language processing based on the use of neural networks. In particular the problem area chosen is learning and correct assembley of words in an imput sentence, allowing in such a way the representation of the meaning of the sentence.
The neural network employed for these experimens is based on the model of 'backward propagation', where learning takes place through weight adjustement among the connections between units, therefore this represents the knowledge which the system employes to achieve corrects words assemblies.

1. INTRODUZIONE

La comprensione del linguaggio naturale da parte di un sistema simulativo richiede come minimo che esso sia in grado di combinare tra loro nel modo corretto i significati delle singole parole che compongono una frase. Ad esempio, data la frase "il cane insegue il gatto", il sistema deve combinare i significati delle tre parole "cane", "insegue" e "gatto" in modo da capire che il cane è l'inseguitore e il gatto l'inseguito, e non viceversa. Lo stesso accade per frasi più complesse, come "il ragazzo che Maria ha conosciuto in Francia le ha scritto che arriverà oggi", dove il sistema deve capire, tra l'altro, che chi arriverà è il ragazzo e non Maria, e che chi ha scritto è di nuovo il ragazzo e non Maria, che la persona a cui il ragazzo ha scritto è Maria, che la Francia è il luogo dove Maria ha conosciuto il ragazzo e non il luogo dove il ragazzo arriverà oggi, e così via.

Se il significato di ogni parola viene rappresentato come una "molecola" formata da un atomo centrale e da uno o più atomi periferici, la comprensione linguistica può essere vista come la soluzione di un problema: quello di stabilire i corretti "legami di assemblaggio" [3] tra coppie di atomi appartenenti a molecole diverse. La soluzione di questo problema consiste nel trovare l'insieme ottimale di legami di assemblaggio, dati una serie di vincoli diversi contenuti all'interno del sistema. Questi vincoli sono relativi all'ordine delle parole nella frase, alla loro morfologia (es. suffissi), alla categoria sintattica di appartenenza e infine alle conoscenze enciclopediche sul mondo possedute dal sistema. Ad esempio, nella frase "il cane insegue il gatto" è l'ordine delle parole che determina gli assemblaggi corretti; nella frase "il cane inseguono i gatti" è la morfologia, mentre nella frase "il tavolo ha visto il ragazzo" sono le conoscenze che il sistema ha sui tavoli, sul vedere e sui i ragazzi.

2. COMPRENSIONE DEL LINGUAGGIO NATURALE

In questo lavoro vogliamo affrontare il problema della comprensione del linguaggio naturale da un punto di vista connessionista [4]. In un modello connessionista i vincoli

che il sistema utilizza sono rappresentati dai pesi posti sulle connessioni tra un grande numero di unità elementari. Questi pesi non vengono dati inizialmente al sistema, ma il sistema li deve "apprendere" attraverso la ripetuta presentazione di un certo numero di frasi e dei loro assemblaggi corretti. Inizialmente il sistema fornisce per ciascuna frase in ingresso degli assemblaggi casuali. Tuttavia, poiché per ogni frase viene dato al sistema anche l'insieme degli assemblaggi casuali. Tuttavia, poiché per ogni frase viene dato al sistema anche l'insieme degli assemblaggi corretti, il sistema può confrontare gli assemblaggi a cui è giunto da solo con quelli corretti e modificare gradualmente i pesi delle connessioni in modo da avvicinare gli assemblaggi computati dalla rete a quelli corretti. Una seconda misura di apprendimento del sistema è la sua capacità di generalizzare il computo degli assemblaggi corretti a frasi nuove, cioè non comprese nell'insieme delle frasi sulle quali si è basato l'apprendimento. Nella seguente sezione descriveremo più in dettaglio il nostro esperimento.

3. IL MODELLO PROPOSTO

Il modello che abbiamo sviluppato è costituito da tre livelli di unità:

a) Un livello di input che contiene delle unità, dette unità di input, che rappresentano le parole che il sistema è in grado di processare e che vengono attivate quando viene presentata una frase al sistema, e delle unità di memoria (Jordan, 1985) che conservano traccia di quello che il sistema ha processato precedentemente.

b) Un livello intermedio che contiene delle unità nascoste (hidden units) che servono al sistema per elaborare l'input.

c) Un livello di output che contiene delle unità che servono al sistema per rappresentare il risultato del processamento dell'input (cioè, nel nostro caso, gli assemblaggi tra gli atomi delle molecole corrispondenti alle parole della frase).

Ogni unità del primo livello (cioè ogni unità di input e ogni unità di memoria) è connessa con tutte le unità del livello intermedio e ogni unità del livello intermedio

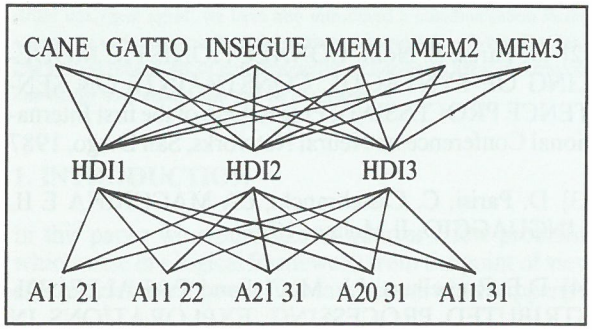


Fig. 1 (per una spiegazione dei simboli usati in questa figura vedi avanti).

(hidden) è connessa con tutte le unità del livello di output. I pesi di tutte queste connessioni sono fissati all'inizio in modo casuale e vengono poi modificati attraverso un processo di apprendimento.

Inoltre, ogni unità del livello intermedio è connessa con una corrispondente unità di memoria del livello di input con un peso pari a 1 che resta fisso (nel senso che non viene modificato durante il processo di apprendimento).

Quando il valore di attivazione di una unità di input viene modificato, l'attivazione si propaga dalle unità del primo livello a quelle del secondo livello e da queste:

a) alle unità del terzo livello così che si produce un output (cioè una modificazione dello stato di attivazione delle unità di output);

b) alle unità di memoria del primo livello così che quando un nuovo input è fornito dall'esterno al sistema il sistema stesso dispone oltre che di tale input (rappresentato nelle unità di input) dell'input costituito dai precedenti stati di attivazione del sistema stesso (rappresentati nelle unità di memoria).

Con l'architettura descritta, in particolare, poiché ogni unità del livello intermedio è connessa con una corrispondente unità di memoria con un peso fisso pari a 1, lo stato di attivazione delle unità di memoria in un dato tempo sarà eguale allo stato di attivazione delle unità del livello intermedio nel tempo precedente (il quale tuttavia a sua volta sarà il risultato dei precedenti stati di attivazione del sistema nei tempi precedenti).

Le unità di input sono rappresentate, per semplicità, in modo localistico; così esiste una unità che rappresenta la parola "cane", una che rappresenta la parola "cani" ecc., e non in modo distribuito (cioè utilizzando più unità per rappresentare una sola parola e lo stesso insieme di unità per rappresentare parole diverse). L'informazione sulla sequenzialità dell'input (cioè sull'ordine delle parole che costituiscono la frase da processare) viene rappresentata temporalmente fornendo in input al sistema una parola alla volta.

Così, ad esempio, quando viene presentata al sistema la frase "il cane dorme" viene attivata prima l'unità di input corrispondente alla parola "il" e viene lasciata propagare l'attivazione; poi l'unità CANE (cioè l'unità corrispondente alla parola "cane") e viene lasciata propagare l'attivazione, e infine l'unità DORME e viene lasciata propagare l'attivazione.

L'output è rappresentato anch'esso, per semplicità, in modo localistico ovvero con una sola unità per ogni possibile assemblaggio. Esiste, ad esempio, una unità di output A11-21 che rappresenta l'assemblaggio tra il primo argomento della molecola corrispondente alla prima parola e il primo argomento della molecola corrispondente alla seconda parola, una unità di output A11-22 che rappresenta l'assemblaggio tra il primo argomento della molecola alla prima parola e il secondo argomento della

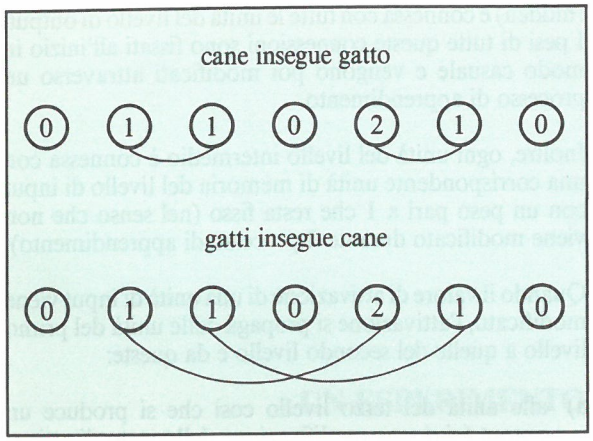


Fig. 2

molecola corrispondente alla seconda parola, ecc. (dove A significa assemblaggio, il primo numero identifica la prima molecola, il secondo numero l'argomento della prima molecola, il terzo numero la seconda molecola e il quarto numero l'argomento di tale seconda molecola).

Così, ad esempio, produrre gli assemblaggi corretti per la frase "cane insegue gatto" (tralasciamo gli articoli per semplicità) significa per il sistema attivare i nodi output A11-21 e A22-31, che rappresentano rispettivamente l'assemblaggio tra il primo argomento della molecola di "cane" (la molecola 1) e il primo argomento della molecola di "insegue" (cioè colui che insegue) e l'assemblaggio tra il secondo argomento della molecola di "insegue" (cioè colui che è inseguito) e il primo argomento della molecola 3, cioè la molecola di "gatto" in questa frase. Mentre, ad esempio, produrre gli assemblaggi corretti per la frase "gatti insegue cane" significa per il sistema attivare i nodi output A11-22 e A21-31. Vedi la fig. 2 dove abbiamo indicato gli argomenti delle molecole con dei cerchi numerati e gli assemblaggi con delle cordicelle.

Le conoscenze che permettono al sistema di produrre gli assemblaggi corretti per una frase presentata in input (cioè le conoscenze che il sistema utilizza per "capire" le frasi) sono rappresentate nei pesi delle connessioni tra le unità e tali pesi sono sviluppati dal sistema attraverso una fase di apprendimento durante la quale esso usa una procedura detta Back Propagation [4].

Tale fase di apprendimento, brevemente, consiste nella presentazione in input al sistema di una lista di frasi e di una corrispondente lista di output desiderati (presentazione che viene ripetuta per un gran numero di volte). Durante questa fase il sistema, oltre ad eseguire la normale procedura di progagazione dell'attivazione in seguito agli input forniti, confronta l'output risultante dal suo stato di pesi corrente (che da principio produrrà un output casuale) con quello richiesto e in proporzione agli errori riscontrati modifica lentamente il peso delle connessioni sino ad ottenere una configurazione di pesi ottimale.

Nel nostro caso durante il processo di apprendimento sono state presentate in input al sistema una serie di frasi molto brevi con associati, per ogni frase, gli assemblaggi desiderati:

"cane insegue gatto" -> A11-21 A22-31
"gatti insegue cane" -> A11-22 A21-31
"cane dorme" -> A11-21
"....." -> .....

Dopo tale fase di apprendimento il sistema risulta in grado di riprodurre gli output corretti (cioè gli assemblaggi giusti) per le frasi presentate durante la fase di apprendimento (cioè quelle frasi per le quali è stato indicato al sistema l'output desiderato) ma soprattutto, in alcuni casi, è in grado di processare correttamente frasi che non ha mai visto prima, come ad esempio le frasi:

"gatto dorme"

"cane insegue cane".

Cioè il sistema ha estratto, dagli esempi fornitigli, delle regolarità valide anche per input diversi da quelli processati durante la fase di apprendimento. Nel caso particolare delle frasi citate possiamo dire che evidentemente il sistema ha imparato che la parola "gatto" e la parola "cane" hanno un comportamento molto simile e dunque trovando una di tali parole in un contesto in cui aveva trovato l'altra ha prodotto lo stesso output che sapeva essere giusto per quella parola.

L'esperimento descritti non è che un tentativo iniziale di costruire una "rete neuronale" capace di apprendere ad assemblare in modo corretto i significati delle singole parole di una frase in ingresso, generando così una rappresentazione complessiva del significato della frase. La rete che si stabilizza alla fine della fase di apprendimento dovrebbe contenere, nei pesi delle connessioni tra le sue unità, una rappresentazione delle diverse conoscenze (vincoli) (Parisi & Nolfi, 1987) che il sistema utilizza per giungere agli assemblaggi corretti.

4. BIBLIOGRAFIA

[1] M.I. Jordan, THE LEARNING OF REPRESENTATIONS FOR SEQUENTIAL PERFORMANCES, Unplished doctoral dissertaction, University of California, San Diego, 1985

[2] D. Parisi, S. Nolfi, CONNECTIONISTIC MODELING OF SYNTACTIC CONSTRAINTS ON SENTENCE PROCESSING, Proceedings of the first International Conference on Neural Networks, San Diego, 1987

[3] D. Parisi, C. Castelfranchi, LA MACCHINA E IL LINGUAGGIO, Il Mulino, 1987

[4] D.E. Rumelhart, J.L. McClelland, PARALLEL DISTRIBUTED PROCESSING: EXPLORATIONS IN THE MICROSTRUCTURE OF COGNITION, Cambridge, MA: MIT Press/Bradford Books, 1986

CAESAR: COOPERATING AGENTS FOR EXPERT SYSTEM ACTIVITY REPRESENTATION

Antonio D'Angelo, Enrico Pagello (*)

Laboratorio di Intelligenza Artificiale

Dipartimento di Matematica e Informatica

Università di Udine

RIASSUNTO

Lo scopo di questo articolo è quello di illustrare uno strumento adatto per la rappresentazione dell'attività di un *sistema intelligente,* basata sulla cooperazione tra agenti indipendenti. Descriveremo pertanto di seguito le caratteristiche principali di un'architettura software adatta a supportare un *'problem solver' distribuito:* abbiamo realizzato il prototipo di un nucleo di un motore d'inferenza che svolge il ruolo del cuore di un singolo agente intelligente; abbiamo anche introdotto un meccanismo di comunicazione tra i vari agenti basati sul paradigma del *message passing.* Oltre a questo verranno illustrati alcuni commenti di carattere generale relativi alla descrizione di un *agente intelligente* mediante appropriati *livelli di astrazione della conoscenza.*

ABSTRACT

The aim of this paper is to illustrate a suitable tool for representing the activity of an *intelligent system* as a cooperation among indipendent agents. We will describe therefore in the following the outlines of a software architecture suitable to support a *distributed problem solver:* we have implemented a prototype of the kernel of an inference engine which acts as the core of the single intelligent agent; we have also introduced a communication facility among the various agents from the *message passing* point of view. Besides, a few general remark about how to describe an intelligent agent in terms of a *knowledge level* abstraction will be illustrated.

1. INTRODUCTION

In this paper we would like to address a few problems which arise in a logical framework from the point of view af *Artificial Intelligence.* We investigate what happens if we choose a logical language to effectively represent the knowledge in an applicative domain, so that some deductive property of the language in used to realize an *Inference Engine,* namely, an interpreter which allows the language to become an effectively usable implementation language for intelligent system.

Looking back to the evolution in designing intelligent system architectures various different approaches can be roughly identified. The Robinson's pioneering work [1] on the *Resolution Principle* allowed, in the late 60's, the implementation of general purpose *theorem provers* which, using very little domain dependent information and few special heuristics besides those of syntactic nature, have the form of *uniform* proof procedures. Their fundamental drawback was the exponential computation time when relevant consequences from complex situations (axiomatized in clausal form of first order predicate calculus) needed to be derived.

In the early 70's, when much hope in improving such systems was lost, there was a spreading belief that resolution type systems could really work in real world applications without some extensive changes in philosophy. On the other hand, the develpment of *Rule-based System* [2] emphasized the need for much more efforts towards less syntactic implementations which use heuristics and user supplied knowledge and which are domain dependent. Even more, many sofisticated implementations of the *Pattern-Matcher* in rule-based systems which make use of the Rete's algorithm [3], look like a **data driven model of computation**. The principal drawback of such implementations is in that they don't distinguish between domain and control rules, causing the matcher to be applied *uni-*

(*) With Istituto di Elettrotecnice ed Elettronica, Università di Padova and Instituto LADSEB of CNR, Padova, Italy.

formly on every rule in the data base. The resulting *fine-grained* data driven model slows seriosly down the computation so that rule interpreters execute only a few amount of cycles for time unit.

Following the architectural principles appeared in the early 80's, a data driven model of computation to execute control rules only, was proposed as most suitable organization of computation whereas domain rules, appearing in clausal form, could be processed by a **controlled theorem prover**. Its implementation can be viewed ad a *shell* containing any search strategy excutable by a special control module, the *inference controller,* which drives the theorem prover by comparing and, possibly, modifying its behaviour continously. Even this approach, however, supplies a data-driven organisation which suffers of a too fine-grained model of computation.

In this spirit we advocate a *course-grained* data driven model of computation by which implementation details about control and representation models about restricted domain knowledge can be encapsulated in that we call a **reasoning agent**. Depending on a particular domain level application, a **communication protocol** [4] among agents should be provided by the end-user to the aim to allow the overall system to be throught as a *distributed problem solver.*

We will describe therefore in the following the outlines of a suitable software architecture we propose to support a distributed problem solver which we called CAESAR *(Cooperating Agents for Expert System Activity Representation).* The CAESAR system is based upon the idea of representing an intelligent system as set of cooperating agents.

We have implemented a prototype of the kernel of an inference engine, whose behaviour could be driven both by the search space strategy stored into the knowledge base and the base beliefs on which the inference engine is applied; it has been implemented as the kernel for an interpreter which, with its associated knowledge base, can viewed as a single intelligent agent: it will be illustrated in sections 3.3, 4.0, 4.1, 4.2 and 4.3.

The communication facility among agents has been outlined within the message passing framework and will be illustrated in sections 3.2 and 4.0.

Besides a few general remarks about how to describe intelligent agents in terms of a knowledge level abstraction will be illustrated in section 2.0 and 2.1.

2. THE ABSTRACT MODEL OF A REASONING AGENT

In clarifying open questions about knowledge representation, Newell [6] introduces the idea of a *knowledege level,* an abstract view of how an intelligent agent should be characterized. By this approach he can describe an agent's knowledge in term of a *competence* notion, as "being a potential for generating action". Thus, the notion of competence doesn't require an agent be aware of the consequences of what he knows, but assumes an agent be merely aware of where those consequences can be examined. Therefore, the *domain of logic* becomes precisely the study of what is *implicit* in what an agent might believe (Levesque [7]).

Henceforth, because our major goal is the construction of agents whose behaviour must be intelligent enough to perform cooperating tasks involving other agents, it is necessary to address a number of issues that concern the interaction of **knowledge** with **actions** and **planning**. An agent must consider that its future states of knowledge can be related to other agents' plans; hence, if it wants to incorporate those plans into its own, then it must also to reason about the knowledge and plans of other agents in appropriate way.

For many years, however, the principles of intelligent system architecture design have largely remained static. These principles included

1 - a *single reasoning agent* incorporating inference engine, knowledge base and user interface,

2 - *monotonic* increase of knowledge,

3 - *low-level* control rules,

4 - *sequential, centralized* flow of inferences.

We want to introduce, [5] and [8], an abstract model of a reasoning agent. By the term **reasoning agent,** we refer to what the basic component of an intelligen system should be by making the following fundamental assumption:

Every agent is characterized by three components:

1. *- a memory storage where knowledge is maintained and update,*

2. *- an inference mechanism with which knowledge is verified and expanded,*

3. *- a communication capability which can be used to exchange information among other (artificial and/or Human) agents.*

In other words, an agent S_i can be characterized by a triplet

$$S_i = < K_i, L_i, M_i >$$

where

1 - K_i is the agent's *chunk of knowledge,*

2 - L_i is the agent's *inference controller,*

3 - M_i is a generalized mechanism of *message passing.*

This somewhat simplified model is very general since it makes no hypothesys about either the structure of the memory or the communication protocol. A more detailed specification of an agent's internal structure, however, depends seriously on the way knowledge is represented and used ("knowledge should be characterized *functionally,* in terms of what it does, not *structurally,* in terms of objects with particular properties and relations", Newell [6]). In other words, whatever symbol system is used to encode agent's knowledge, agents need to process their knowledge if they want *to feel active* and exhibit some desired behaviour.

The agent's knowledge looks like the standard notion of an abstract data type with the following two primitive operations that, in according to Levesque [7], wa cell ASK and TELL:

— ASK allows an agent to access knowledge by deducing answers from new questions,

— TELL allows an agent to enrich his knowledge by learning new assertions about the world.

The key feature of this approach is that it encapsulates an agent's internal structure in the implementaion of such operations.

3. CAESAR SYSTEM OVERVIEW

Like many other proposed multiagent architectures, the design of the CAESAR system [8] has been motivated by the desire of supplying the end-user with a suitable *programming environment,* where intelligent systems can be developped by implementing a number of indipendent submodules (**reasoning agents**) whose interaction can be defined by appropriate protocols (**mutual knowledge** and **actions**). We have structured the single reasoning agent into several linguistic levels:

— **HRN** *(HORN Representatio Network)* is the low-level inference language for accessing the logical features,

— **KCL** *(Kernel Command Language)* is a low-level set of system call for providing a direct control on the inference processor,

— **QDL** *(Query Definition Language)* is an user-oriented declarative language for interrogating the inference processor in high-level manner,

— **ACT** in an *Agent Cooperation Task* descriptor for representing the manual knowledge and actions among agents.

Therefore, the CAESAR system has been built upon the prototype implementation of what we called the HORN module [5], which realizes single agent's flow of reasoniung. The Linguistic levels in HORN are articulated as in fig. 1.

3.1 The multiagent system

Over the last few years a number of new intelligent system architectures, based upon a decentralized approach to problem solving have been proposed and some experimental distributed systems have been built. The driving force for these architectures stems from the observation that "a *single line of reasoning* is too weak" (Stefik [9]) expecially if it is carried out a single intelligent agent, operating in a relatively static domain under the simplified assumption of the absence of other intelligence agents.

On the other hand, although many proposed multiagent architectures have been motivated by the desire of exploit concurrency to increase problem solving performance, severe restrictions weaken their applicability. So, many distributed problem solvers make the assumption that their component agents have common and not conflicting goals. Such architectures are centered on the question of how these agents can best help one another in carrying out their common task (McArthur and Cammarata [10], Corkill and Lesser [11], Davis and Smith [12]), or how they can avoid interferences while using common resources (Georgeff [13]). In other cases, more relaxed conditions can require intelligent agents either capable of interacting with not identical goals (Fox [14], Yang [15]) or dealing with situations where the complexity of interaction is better handled by a framework that recognized and resolves sub-goal conflicts in a general manner (Rosenschein and Genesereth [16]).

Various architectural principles will contribute to construct a reasoning agent whose behaviour will intelligent

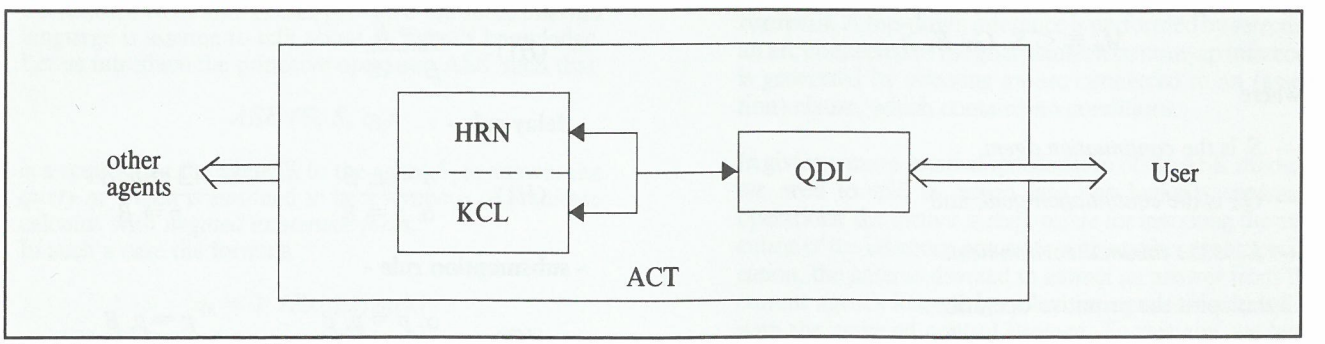


Fig. 1 System Language Hierarchy.

enough to perform cooperating tasks involving other agents: active intelligent agents need to be able to represent the knowledge about how their mutual actions can modify the environment of their interactions. Hence

> *reasoning about protocols among knowledge based is crucial in understanding distributed system behaviour, since* **messages** *can be viewed as changing the state of knowlege of a* **distributed problem solver.**

So far we have argued that a way to realize a reasoning can be carried out by implementing two primitive operations (ASK and TELL) as soon as a suitable representation for an agent's knowledge has been chosen; moreover, a general framework for modelling knowledge needs a more precise characterization of how an agent's knowledge can be used by itself or by other agents.

Thus, following Konolige [17], we shall make three fundamental assumption about the capability of an agent to draw conclusions from its own knowledge; namely,

1 - **resource incompleteness,** which stems from a physical limitation in time and space of any computational device on which the ASK operation is implemented;

2 - **logical incompleteness,** which can arise when an agent lacks an adequate inferential capability because of its incomplete knowledge;

3 - **relevance incompleteness,** which considers an agent's incomplete derivational process because of its erroneus focusing of the problem at the hand.

We shall try in the following to define the structure of an agent's knowledge in the light of the considerations previously outlined. What we want to do is to model an agent's beliefs about the world, including other agents like itself, by using a set of statements in some logical language in a manner that will make reasonably implementable the agent's behaviour. Having identified which properties an agent should incorporate in order to show an intelligent behaviour, the next step is to characterize it functionally.

3.2 The message passing mechanism

As we already mentioned in the section 2, active intelligent agents need to be able to represent the knowledge about how their mutual actions can modify the environment of their intercations. Let us define a **mesage** between two agents a triplet

$$M_{ik} \equiv < S_j, G_{ik}, E_k >$$

where

— S_j is the *continuation agent,*

— G_{ik} is the *continuation goal,* and

— E_k is the *minimal environment.*

In this spirit the primitive operation

$$SEND\ (S_j, S_j, G_{ik}, E_k)$$

realizes a message passing when a request from the agent S_j invokes the activation of the agent S_j under specified conditions.

We must provide our system with a collection of information stored in what we have called **Partition Activation Driver** (PAD) which causes the inference processor to be driven by the strategy codified in the associated partition as planning rules (or, local metarules) which, in turn, can be loosely controlled by other *remote* metarules (local to other agents). The SEND operation controls the execution of the PAD module which, in turn, is responsable to schedule the activities of the inference processor among different agent's knowledge structures.

3.3 Inference control mechanism

Despite how many possible symbolic systems are suited encode knowledge, we have used *clausal form* of logic as knowledge representation because of its convenience for expressing certain classes of problems requiring deductive reasoning in high level feature. The inference processor, which represents the kernel of the single agent architecture, has been designed to the aim of improving efficiency in carrying out inferences; it executes a HRN language representation of a set of clauses *(agent's chunk of knowledge)* as soon as a goal clause *(query)* has been given.

As an inference mechanism we propose a special device that we have called **resolution planning** which allows the inference processor to be driven by a **control strategy** stored itself in a connection graph. The main feature of our approach is to recognize, during the execution of the goal, whether a subgoal can be delayed or when a clause, whose head matches with that subgoal, can be removed from the set of all potentially unifiable clauses.

We use the symbol $\leftarrow$ for the **reverse conditional** as usual (cfr. Kowalski [18]), while the connective $\Rightarrow$ defines a symmetrical binary relation between α and β such that.

> *if all active subgoals β are satisfable, the global goal represented by $\alpha \Rightarrow \beta$ is satisfable provided that all conditioning subgoal α can be proved satisfable.*

The implementation of the *inference controller* is supported by the set of rules listed below.

- exchange rule -

$$(H1) \quad \frac{\alpha \Rightarrow \beta}{\beta \Rightarrow \alpha}$$

- delay rule -

$$(H1) \quad \frac{\alpha \Rightarrow p,\ \beta}{\alpha,\ p \Rightarrow \beta} \qquad \frac{\square \Rightarrow p,\ \beta}{p \Rightarrow \beta}$$

- subsumption rule -

$$(H3) \quad \frac{\alpha,\ p \Rightarrow p,\ \beta}{\alpha \Rightarrow p,\ \beta} \qquad \frac{p \Rightarrow p,\ \beta}{\square \Rightarrow p,\ \beta}$$

- expansion rule -

$$(H4) \quad \frac{\alpha \Rightarrow q, \beta \quad q \leftarrow p_1, \ldots, p_n}{\alpha \Rightarrow p_1, \ldots, p_n, \beta}$$

- elimination rule -

$$(H5) \quad \frac{\alpha \Rightarrow p, \beta \quad p}{\alpha \Rightarrow \beta} \qquad \frac{\alpha \Rightarrow p \quad p}{\square \Rightarrow \alpha}$$

- termination rule -

$$(H6) \quad \frac{\square \Rightarrow p, \quad p}{\square}$$

- relevance rule -

$$(H7) \quad \frac{R_i \sim q \quad <R_i, q \leftarrow p_1, \ldots, p_n>}{q \leftarrow p_1, \ldots, p_n}$$

$$\frac{F_i \sim p \quad <F_i, p`>}{p`}$$

In other words, the relation $\alpha \Rightarrow \beta$ splits the sequence of goals α, β into two classes with the important property that delayed subgoals α are the condition in order to satisfy the global goal α, β when all active subgoals β have already been satisfied. Moreover, the symbol $\sim$ defines a *relevance relation* between a subgoal and the possibly related rules and/or facts in a sense that it can be explicitally estblished which rules or facts should be considered relevants for the current problem solving.

At this point we may introduce the function specification of ANSWER

$$ANSWER\ (L_i, \alpha, \beta, E_k)$$

with the specification that, under the local control L_i, tha agent tries to solve the *conditioned goal* $\alpha \Rightarrow \beta$ in the environment E_k.

4. HORN MODULE ARCHITECTURE

As we have argued in the preceding sections a reasoning agent can be functionally characterized by two primitive operations, ASK and TELL, provided tha some *internal* language is assume to talk about an agent's knowledge. Let us introduce the primitive operation ASK such that

$$ASK\ (S_i, S_j, q_k)$$

is a request for the agent S_i to the agent S_j to answer the *query* q_k which is assumed to be a sentence of predicate calculus with *negated existential form.*
In such a case the formula

$$q_k \equiv \not\exists\ (Exist\ x)\phi_k(x)$$

allows to interpret the query as a negated clause which activates the inference processor, included in the HORN module, on the knowledge base K_i with the local control L_j of the agent S_j.

The overal structure of a single reasoning agent is shown in fig. 2, where the ASK module implements either a **deductive reasoning** on the knowledge base K_i with given control L_i or a **message passing** to another agent S_l.

The CONSULT, CLASSIFY and TELL primitives are used by the agent S_i to handle appropriately his knowledge base, which is implemented as a triplet

$$K_i \equiv <D_i, T_i, B_i>$$

where

1 - D_i is the **dictionary** of all the terms known to the agent S_i, included *properties* about objects, relations, facts and rules and which all identified by a name appearing in the dictionary;

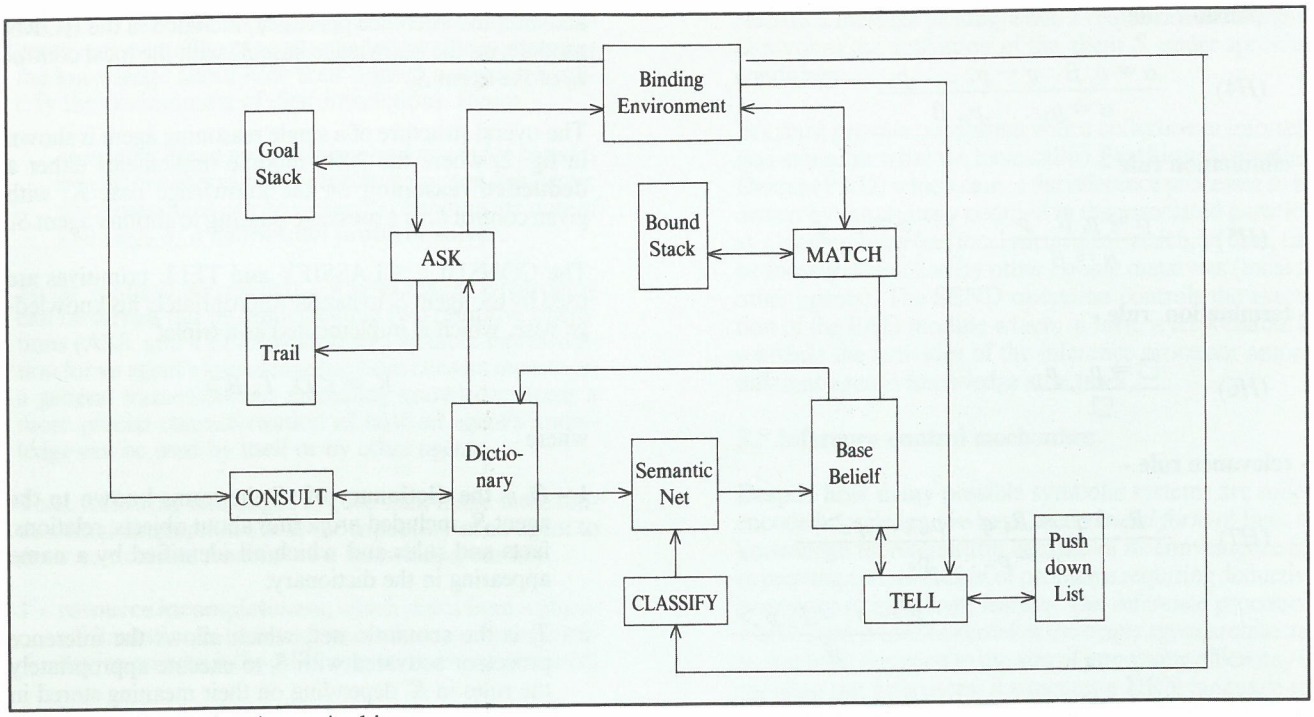
2 - T_i is the **semantic net,** which allows the inference processor activated with S_i to execute appropriately the rules in K_i depending on their meaning stored in T_i;

3 - B_i is the agent's **base belief** about the world, including rules and fact about rules, namely, the agent's metaknowledge.

Two main dynamic areas are used by the inference to perform deductions on K_i, namely the **GoalStack** H_i and the binding **Environment** E_i. the former organized as a bidirectional stack with on side containing all active current subgoals and the other eventually used to push down delayed sudgoal descriptors; on the contrary, the latter is arranged as *heap* and it is used to mantain all generated variable-bindings during unification (performed by the MATCH submodule) until any backtracking request doesn't cause to clear some already bound variable. The secondary stack ***trail*** is used only as a *reset list* during backtracking and it is not further considered.

The basic operation of the inference processor is the selection of an arc and the generation of the associated resolvent which, in turn, is temporary added to the graph by allocating a new *clause descriptor* to the top of the **GoalStack** and a new *binding descriptor* to the **Environment.** A top-down inference is performed by selecting an arc connected to the goal clause; a bottom-up inference is generated by selecting an arc connected to an (assertion) clause, which contains no conditions.

In giving a more detailed specification of the ASK module, we need to split its action into two logically separated operations: the former is responsible for invoiking the execution of the inference processor with another agent specification, the latter is devoted to extract an answer from the current agent's knowledge when an answer is admissible with the assigned control strategy. To this aim we have introduced the following two operators as stated above.



Fi. 2 Single Reasoning Agent Architecture

In givin a more detailed specification af the ASK module, we need to split its action into two logically separated operations: the former is responsable for invoiking the execution of the inference processor with another specification, the latter is devoted to extract an answer from the current agent's knowledge when an answer is admissible with the assigned control strategy. To this aim we have introduced the following two operators as stated above

$$SEND\ (S_i, S_j, G_{ik}, E_k)$$

$$ANSWER\ (L_i, nil, G_k, E_k)$$

where the former implements a message passing mechanism while the represents the answer the given goal G_k with the related environment E_k under the assigned control strategy L_i. Again, two minor stacks are supplied locally with the specialized modules MATCH and TELL, whose implementation has been designed in iterative form for a reason of efficiency.

4.1 The knowledge base organisation

In the preceding sections we have seen that the knowledge base for a single agent S_i is organized as a triplet

<dictionary, semantic net, base belief>

whose overall implementation is **a connection graph** for the current agent's knowledge. In our implementation every term in the dictionary has one more *entry-points* in the semantic net which, in turn, allows a *selective acces* to the data base of the rules and facts known to the current agent.

From a technical point of view, the dictionary has been implemented by means of three tables, namely, *SkelTable, AtomTable* and *HashTable* where the second has been merely used to store the characters strings associated with every term whereas the last would allow a direct access to the dictionary by the means of a *hash* function.

The structure of the dictionary, however, has been represented by the first table, whose elements are named **skeletons** (or, *Skel*) and whose attributes represent all the information needed for a correct handling of the terms in the dictionary. The abstract structure of each term is more exactly a **functor** which is characterized by a name and the number of its parameters.

<Skel> ⇒ <Next><Name><NVar><Gndx><Form>
<Next> ⇒ <Skel>
<Name> ⇒ <Char> | <Char><Char>
<NVar> ⇒ <integer>
<Form> ⇒ <Skel> | <Skel><Vars>
<Vars> ⇒ <IVar> | <IVar><Vars>
<Char> ⇒ <printable character>
<IVar> ⇒ #<integer>

where

Next -represents a reference to the following skeleton in the chained subdictionary indentified by the same entry-point hash value;

Name -allows the inference processor a fast access to the codified string which represent the name of the functor;

NVar -specifies the number of arguments for the current functor;

Gndx -provides an entry-point in the semantic net in according to some defined characteristics specified by the user for the current functor;

Form -defines a reference for the constructed term, created during the generation of the term in the dictionary, and used by *Unify.*

Moreover, for every term, the semantic net can be accessed by a couple of entry-points depending whether the current term is identified either as a subgoal, in which case inferences are carried out in top-down manner, or as an assertion, in which case inferences are carried out in bottom-up manner.

The connection graph generated by the **semantic network** provides a labelling method between terms in the dictionary and the referenced terms appearing in the facts and the rules representing the base belief for a single agent. By such an implementation strategy we can assume that the size of the semantic net can be interpreted as a measure of domain specific information an agent knows. In other words, the interpretation of *axioms*, which represent an agent's knowledge and are codified in clausal form, drives the inference processor towards the selection of the most promising or fact to resolve upon the already selected subgoal. The semantic net can be generated by the following BNF grammar

<*Gndx*> ⇒ <*Backward*><*Forward*>
<*Backward*> ⇒ **GOAL**<*Entry*>
<*Forward*> ⇒ **DATE**<*Entry*>
<*Entry*> ⇒ **ISA**<*Entry*><*Entry*>
<*Entry*> ⇒ **PARTOFF**<*Entry*><*Entry*>
<*Entry*> ⇒ **INSTOF**<*Entry*><*Entry*>
<*Entry*> ⇒ <*Rule*> | <*Fact*>

where *Goal* and *Data* are used distinguish between top-down and bottm-up inferences, while *IsA, PartOf,* and *InstOf* denote the usual constructors appearing in litiratura [19] to implement semantic network. A similar idea was segguested by Kowalski [20] where an *indexing scheme* has been introduced for guiding the search for a solution so that the semantic network can be viewed also as an abstract data type to implement a proof procedure.

Finally, the knowledge an agent S_i has about the world is stored in the so called base beliefs B_i associated with the agent and whose implementation has been achieved by a couple of data structures, namely, *ClauseTable* and *Memory.* The latter is arranged to maintain the agent's base beliefs as a collection of facts and rules codified in the clausal form of logic, whereas the former holds a brief description for every fact or rule appearing in the latter allowing to refer to it as an atomic unit. Also in this case the base beliefs can be generated by the following BNF grammar.

<*Entry*> ⇒ <*Clause*>
<*Rule*> ⇒ <*Clause*>
<*Clause*> ⇒ <*Class*><*NVar*>*its*>
<*Class*> ⇒ <*Skal* >
<*NVar*> ⇒ <*integer*>
<*Lits*> ⇒ **CONS** <*Literal* >*Lits*> | **NIL**
<*Literal* > ⇒ <*Mode*><*Term*>
<*Mode*> ⇒ **PLUS** | **MINUS**
<*Term*> ⇒ <*Skel*> | <*Skel* >*Parms*>
<*Parms*> ⇒ <*Argument*> | <*Argument*>*Parms*>
<*Argument*> ⇒ <*Variable*> | <*Term*>
<*Variable*> ⇒ #<*integer*>

where *Class* characterizes the type of clause under consideration, namely, provides a reference in the dictionary, whereas *Lits* identifies the list of all literals appearing in a clause, codified in *Memory.*

4.1 The inference processor

The inference processor, which represents the kernel of the agent architecture has been built upon an experimental implementation; since it has been designed to the aim of improving effieciency in carrying out inferences, an *inference controller* has been introduced to drive inferences. At present, however, the ASK module only implements the primitive operation ANSWER $(L_i, \text{nil}, G_k, E_k)$ with L_i realizing a *depth-first* search mechanism as in standard PROLOG implementations.

The operation of the ANSWER module is achieved by a data-driven model of computation which controls the execution of inferences in according to the strategy codified as *condition-act* rules defining a particular inference engine. The processor is characterized by a number of **microstates** each of which has been with the values taken by some associated *Flags* and the contents of the *top* of the *control stack* $\mathbf{Q}_i$. Using such a mechanism, a way to select a new microstate of the processor can be defined as soon as the code for the current microstate has been executed. If the microcode is modified by channing the kind of control L_i, then also the operation of the ANSWER module is modified.

The implementation of top-down control with a depth-first strategy has been realized by introducing the following states

S0 - **Arrive** makes use of the current subgoal to enter the dictionary with the associated term;

S1 - **Restore** implements *backtracking;*

S2 - **Enter** allows to select a path in the semantic net (at the presente realized as two-level selection tree) whose leavels refer to all the clauses whose head is potentially unifiable with the current subgoal;

S3 - **Choice** builds explicity an OR node during the generation of the search space;

S4 - **Call** pushes on the top of the *GoalStack* the descriptor of the selected clause to resolve upon the current subgoal;

S5 - **Match** performs *unification;*

S6 - **Eval** implements the primitive operations built in the system and whose description appears in the next section;

S7 - **Proceed** continues towards the next subgoal at an AND node of the generated search space;

S8 - **Return** goes up to some remote, already active, subgoals.

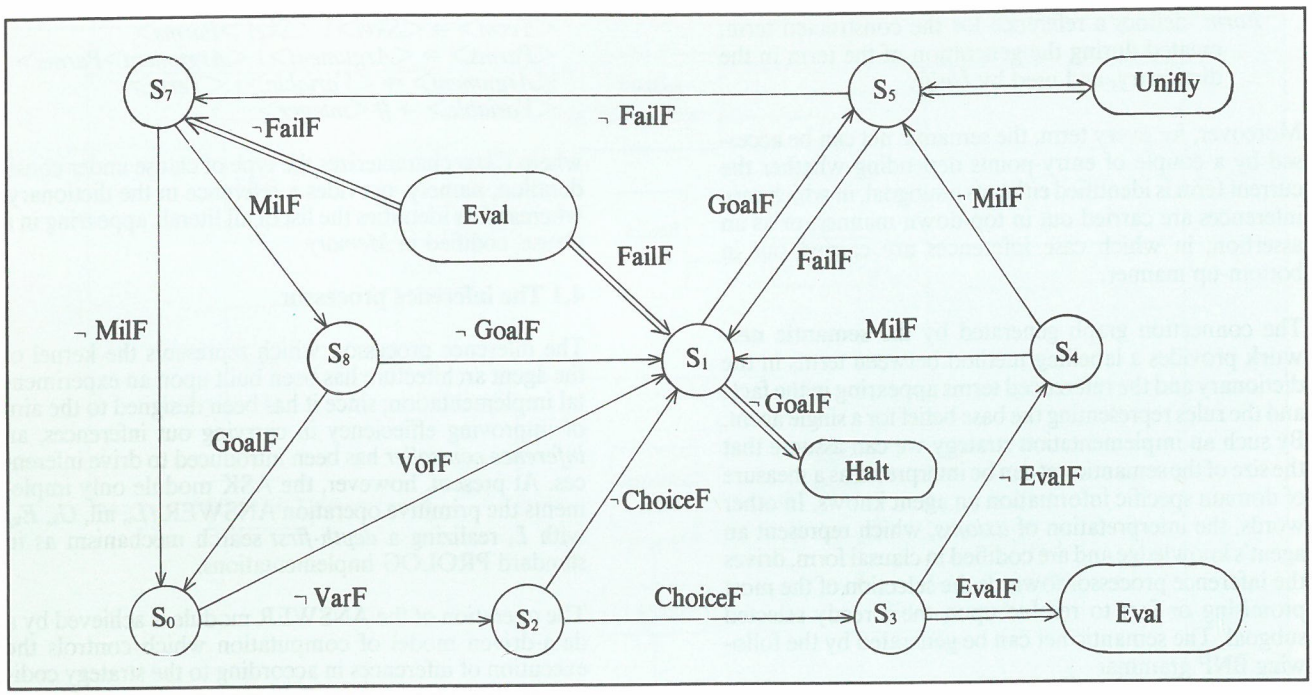


Fig. 3 State Transition Diagram

On the other hand, a depth-first strategy requires the following state *Flags:*

F0 : **VarF,** to advise wherther a selected subgoal can be dereferenced into a term of the dictionary;

F1 : **GoalF,** to signal some subgoals pending at an AND node of the generated search spece;

F2 : **ChoiceF,** to single out, by means of the semantic net, wherther some clause can be resolved upon the current subgoal;

F3 : **CallF,** to signal that the *CallFrame* of the selected clause has been built;

F4 : **EvalF,** to advise that the current subgoal has been associated with a code to be executed;

F5 : **FailF,** to notify either an unification failure or a code execution failure;

F6 : **NilF,** to detect if the current pool of subgoals is empty.

The characterizatione of an inference control L_i as a depth-first search space mechanism [21] has been achieved by committing to the microstate execution code some condition to be tested. As you can see in fig. 3, where a state transition diagram is shown for the ABSWER module under consideration, each standard inferential step is characterized by the following sequence of microstates

$$\textit{Arrive-Enter-Choice-} \left\{ \begin{matrix} \textit{Call-Match} \\ \textit{Eval} \end{matrix} \right\} - \left\{ \begin{matrix} \textit{Proceed} \\ \textit{Return} \end{matrix} \right\}$$

where every different situation is treated as abnormal, causing the processor to discard the last generated search space by means of *backtracking* since it is assumed not promising for the overall solution of a given goal.

You can see, however, that an uniform handling of every abnormal situation ha as consequence a blind search in an AND/OR search space since a different order of execution for a group of AND subgoals can cause a *backtracking,* namely, crossing the state S_i. More intelligent search strategies, relaxing the constraint of subgoal sequences by the introduction of auxiliar microstates, as **Delay** and **Change,** allow a more flaxible control with less overhead due to a backtracking heavy use.

Further microstates, as **Remove,** should provide a more constructive use of OR node, which are entry points in the semantic net and which can allow a path deletion during the generation of the search space. in such a way incomplete search strategies can be appropriately handled.

4.3 The inference processor language

The inference processor implementation provides a low-level base language to access directly the inference mechanism realized by the ASK module which, at present, is implmented only by the means of the ANSWER submodule. A BNF grammar for the sequence of literals appearing in *Memory* in the same as the grammar specified for *<List>* with some obvious modifications which have been needed for the actual implementation of an agent's base beliefs. You can notice, however, that the KCL language provides a direct access to the primitive operations which can be executed by the inference processor and which cen be classified into the following six classes:

1. - **Input/Output Operations,** which allow the inference processor either to be connected to a console or any other specified peripheral device *(GetC, Skip, PutC, Line, Output)* or to control the type of access *(Echo, Tty, Connect);*

2. - **Logical and Arithmetic Operations,** which define the arithmetic standard operations *(Plus, Minus, Mult, Div, Mod, Leq, Lss, ...);*

3. - **Dictionary Handling,** which allows the construction and the to any term in the dictionary *(Skel, Equal, Letter, Digit, Atom, Var);*

4. - **GoalStack Control,** to access directly the GoalFrame of any specified subgoal *(/, Cut, Ancestor, Change, Delay);*

5. - Knowledge Base Access, to construct and maintain the base of rules and facts *(Insert, Where, Delete, Remove, IsA, InstOf, PartOf);*

6. - **Dumping Operations,** to access information during and at the and of execution of any specified sequence of subgoals *(Status, Show, Save, Debug, GetAtrb).*

5. CONCLUSIONS

This paper has proposed how to structure a software architecture which allows to implement an intelligent system as a collection of reasoning agents, each one specialized in dealing with circumscripted problems. In such a way the global behaviour of the system can be programmed by introducing an appropriate communication protocol depending on the current problem at the hand. We expect this model of distributed inference processing to be a very powerful mechanism for controlling access to shared data in modular representation of knowledge and action.

At present, only a partial prototype of the HORN module has been realized using PASCAL as an implementation language. In particular both the HRN and the KCL, language interpreters run on a access to shared data in modular representation of knowledge and action.

At present only a partial prototype of the HORN module has been realized using PASCAL as an implementation language. In particular both the HRN and the KCL language interpreters run on a Sun-Workstation. A general sketch only, of the QDL, language does exist: we forecast to code the **incore** compiler of QDL in QDL itself, so that the system will preprocess QDL sentences to produce the related HRN graph-structure to be stored. No complete specification for ACT has been given yet.

A medical application is under developping in the field of *Problem-oriented Medical Record* and whose major goal is the characterization of a **Medical Assistant** in compiling clinical data records [22].

6. ACKNOWLEDGEMENT

The research was partially supported by a grant MPI 40%, Artificial Intelligen Project *Architetture Software per Sistemi Intelligenti.*

7. REFERENCES

[1] J.A. Robinson, A MACHINE-ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE, J. ACM 12, 1965

[2] S.J. Rosenschein, THE PRODUCTION SYSTEM: ARCHITECTURE AND ABSTRACTION, appeared in Pattern-Directed Inference System, Academic Press, London (England), 1978

[3] C.L. Forgy, RETE: A FAST ALGORITM FOR THE MANY PATTERN / MANY OBJECT PATTERN MATCH PROBLEM, Artificial Intelligence 19, 1982

[4] C. Hewitt, A UNIVERSAL MODULAR ACTOR FORMALISM FOR ARTIFICIAL INTELLIGENCE, IJCAI-73, Stanford (California), 1973

[5] A. D'Angelo, HORN: AN INFERENCE ENGINE PROTOTYPE TO IMPLEMENT INTELLIGENT SYSTEM, International Workshop on Foundation of Logic and Functional Programming, Trento (Italy), 1986, to appear in Lecture Notes of Computer Science, Springer-Verlag

[6] A. Newell, THE KNOWLEDGE LEVEL, Artificial Intelligence 18, 1982

[7] H.J. Levesque, FOUNDATIONS OF FUNCTIONAL APPROACH TO THE KNOWLEDGE REPRESENTATION, Artificial Intelligence 23, 1984

[8] A. D'Angelo, PIGPIE: A SOFTWARE ARCHITECTURAL SUPPORT FOR THE CAESAR SYSTEM, The International Conference on Information Technologies for Development, Bled (Yugoslavia), 1986

[9] M. Stefik, et al., THE ORGANISATION OF EXPERT SYSTEM, A TUTORIAL, Artificial Intelligence 18, 1982

[10] S. Cammarata, D. McArthur, R. Steb, STRATEGIES OF COOPERATION IN DISTRIBUTED PROBLEM SOLVING, IJCAI-83, Karlsruhe (WG), 1983

[11] D.D. Corkill, V.R. Lesser, THE USE OF META-LEVEL CONTROL FOR COORDINATION IN A DISTRIBUTED PROBLEM SOLVING NETWORK, IJCAI-83, Karlsruhe (WG), 1983

[12] R. Davis, R.G. Smith, NEGOTATION AS A METAPHOR FOR DISTRIBUTED PROBLEM SOLVING, Artificial Intelligence 20, 1983

[13] M. Georgeff, A THEORY OF ACTION FOR MULTIAGENT PLANNING, AAAI-84, Austin (Texas), 1984

[14] M.S. Fox, AN ORGANITATIONAL VIEW OF DISTRIBUTED SYSTEMS, IEEE-SMC-11 (1), 1981

[15] D. Yang, et al., AN ARCHITECTURE FOR CONTROL AND COMMUNICATIONS IN DISTRIBUTED ARTIFICIAL INTELLIGENCE SYSTEMS, IEEE-SMC-15, 1985

[16] J.S. Rosenschein, M.R. Genesereth, DEALS AMONG RATIONAL AGENTS, IJCAI-85, Los Angeles (California), 1985

[17] K. Konolige, BELIEF AND INCOMPLETENESS, SRI-TN-319, SRI International, Menlo Park (CA), 1984

[18] R.A. Kowalski, PREDICATE LOGIC AS PROGRAMMING LANGUAGE, IFIP-74, Stockolm (Sweden), 1974

[19] L.K. Shubert, EXTENDING THE EXPRESSIVE POWER OF SEMANTIC NETWORK, Artificial Intelligence 7, 1976

[20] A. Deliyanni, R.A. Kowalski, LOGIC AND SEMANTIC NETWORKS, Comm. of ACM, 22 (3), 1979

[21] R.A. Kowalski, D. Kuehner, LINEAR RESOLUTION WITH SELECTION FUNCTION, Artificial Intelligence 2, 1971

[22] A. D'Angelo, F. Noventa, A. Volpe, CORAL: AN A.I. APPROACH TO THE PROBLEM-ORIENTED MEDICAL RECORD, MedInfo-85, Helsinki (Finland), 1985

LA BASE DI CONOSCENZA DI CHAOS: LA STRUTTURA ED IL LESSICO DELL'UFFICIO

Alfonso Gerevini, Sandro Bottarelli

Dipartimento di Scienze dell'Informazione

Università di Milano

RIASSUNTO

In questo articolo vengono presentati gli sviluppi di un sistema per l'automazione d'ufficio, Chaos II (Commitment Handling Active Office System), che è in grado di gestire conversazioni tra i membri di una organizzazione. Tali sviluppi riguardano:

— La caratterizzazione dei meccanismi di strutturazione dell'organizzazione in aree di responsabilità ed esperienza, delle modalità di attribuzione e revoca di responsabilità e di esercizio di quest'ultima nelle conversazioni.
— L'interpretazione delle espressioni in linguaggio naturale che specificano il contenuto proporzionale delle conversazioni.

ABSTRACT

This paper presents the most recent issues in the development of Chaos II (Commitment Handling Active Office System). Chaos II is an Office Automation System designed to handle conversations inside an organization. These progresses concern the following topics:

— How a human organization gets structured in subareas.
— How subareas management is organized.
— How specialized skills are distributed over subareas.
— How management responsibility is distributed.
— How subareas management and skills enter in converfsation inside the organization.
— How natural language expressions are interpreted.

1. INTRODUZIONE

Chaos (Commitments Handling Active Office Systems) è il nome di una famiglia di sistemi basati sulla conoscenza, in grado di gestire la rete delle conversazioni tra i membri di un ufficio, ed in particolare la rete di impegni da loro presi durante tali conversazioni.

Il progetto è attualmente in via di sviluppo presso il Dipartimento di Scienze dell'Informazione dell'Università di Milano ad opera di un gruppo di ricerca di cui fanno parte F. De Cindio, G. De Michelis, G. Ferrari, C. Simone, R. Vassallo, A. Zanaboni e gli autori di questo articolo.

Il progetto Chaos prese inizialmente spunto dal package COORDINATOR, sviluppato dalla Action Technology [22] sulla base della Teoria degli Atti Linguistici di J. Searle [12] adattato ed utilizzato da F. Flores [6] per analizzare i processi di comunicazione che avvengono nelle organizzazioni e quindi per costruire e coordinare reti di conversazioni nel contesto di sistemi di Comunicazione elettronico.

Due tesi discusse nell'anno accademico 84/85 ([15], [21]) hanno assunto tale approccio per costruire un primo prototipo, Chaos1. L'implementazione di Chaos1 ha dimostrato la fattibilità del progetto, cioè la possibilità di sviluppare strumenti di Automazione d'Ufficio con capacità di costruire rappresentazioni delle organizzazioni sulla base degli impegni vigenti e con capacità di acquisire, dalla gestione delle conversazioni, conoscenza sull'evoluzione dell'organizzazione. Si tratta cioè di strumenti in grado di offrire, all'utente finale, un Supporto intelligente ed adattativo per coordinare il suo lavoro con gli altri membri dell'organizzazione.

In questo articolo vengono presentati gli sviluppi del progetto per ciò che riguarda due aspetti che distinguono Chaos2 rispetto al primo prototipo:

— La caratterizzazione dei meccanismi di strutturazione dell'organizzazione in aree di responsabilità ed esperienza, delle modalità di attribuzione e revoca di responsabilità e di esercizio di quest'ultimo nelle conversazioni.

— L'interpretazione del linguaggio utilizzato nelle conversazioni al fine di fornire agli utenti da una parte la possibilità di utilizzare nei propri messaggi un linguaggio espressivo e flessibile quanto quello naturale, dall'altro un supporto nella prevenzione di possibili fonti di incomprensione o fraintendimento che l'utilizzo di espressioni in linguaggio naturale può generare tra gli interlocutori della conversazione.

1.1 Presupposti teorici di Chaos

Il background che sottostà alla tecnologia proposta può essere sommarizzato come segue:

— Un'organizzazione sociale è un sistema chiuso ed autonomo [10], [5];

— nelle organizzazioni sociali le persone giocano un gioco linguistico [20] le cui regole caratterizzano completamente le persone come appartenenti ad una specifica organizzazione [4];

— negli uffici le persone comunicano per assumere reciprocamente impegni, o per aprire spazi di possibilità per le azioni future [6], [17];

— il computer è un mezzo di comunicazione che disciplina le comunicazioni uomo-uomo in contesti pragmatici [3].

Le espressioni linguistiche non sono eventi individuali non correlati, ma avvengono in strutture conversazionali. Una conversazioni tra due interlocutori è costituita da una sequenza di Atti Linguistici correlati, che possono essere classificati sulla base della Teoria degli Atti Linguistici nei termini del loro contenuto illocutivo. Il contenuto illocutivo del primo atto linguistico che occorre in una conversazione (Richiesta, Offerta, ecc.) caratterizza il tipo di impegno generato dalla conversazione, e il suo contenuto proposizionale caratterizza l'azione oggetto della conversazione.

In accordo con [6], [19] e [17] all'interno di un ufficio esistono vari tipi di conversazioni. Tra questi, due rivestono particolare importanza: le Conversazioni per **l'Azione** e le **Conversazioni per le Possibilità.**

Le Conversazioni per l'Azione sono caratterizzate dalla definizione di un impegno per quel che concerne lo svolgimento di un'azione. Chaos è di supporto a due tipi di Coversazione per l'Azione: le conversazioni aperte da una Richiesta, in cui la persona che apre la conversazione chiede all'interlocutore la presa di un impegno riguardo ad una certa azione e le conversazioni aperte da una Offerta (Promessa), in cui la persona che apre la conversazione si offre per assumersi un certo impegno.

Le Conversazioni per Possibilità sono, in Chaos, le conversazioni in cui i due interlocutori discutono riguardo a nuovi domini di possibilità, e vengono concluse con dichiarazioni di delega di responsabilità, di apertura di nuove aree di attività, o dichiarazioni che ridefiniscono le relazioni che individuano la struttura organizzativa dell'ufficio.

In Chaos2 esiste una tipologia definita di Conversazioni per Possibilità (dichiarazioni di delega di responsabilità, di revoca, di ristrutturazione, ecc.) assente in COORDINATOR nel quale le Conversazioni per Possibilità sono conversazioni "libere", in cui gli interlocutori discutono scambiandosi messaggi reciproci.

1.2 Il sistema Chaos2

In figura 1 è schematizzata un'architettura di massima di Chaos2. Il sistema è costituito da quattro componenti. Come in Chaos1 [16], il **Modulo Gestore delle Conversazioni** si occupa della spedizione, ricezione, memorizzazione degli atti linguistici e della ricerca delle conversazioni. Il **Modulo di Interfaccia** consente all'utente di eseguire atti linguistici, selezionare le conversazioni ed eseguire operazioni come la visualizzazione delle conversazioni (ad esempio quelle per le quali è scaduto il tempo fissato per il completamento dell'impegno).

Il mondo di Chaos2 è l'organizzazione e tutto ciò che avviene in esso sono conversazioni tra i membri dell'organizzazione. In questa ottica le conoscenze di Chaos2 devono riguardare la struttura dell'organizzazione, la rete degli impegni tra i suoi membri e le conoscenze linguistiche che incorporano il modello del dominio di applicazione in cui si svolgono le attività dell'ufficio. Le conversazioni tra i membri dell'organizzazione sono inoltre il mezzo con cui Chaos2 apprende nuove conoscenze.

Le due componenti del sistema, che gestiscono tale conoscenza sono:

— il **Modulo Esperto** dell'organizzazione, che gestisce la conoscenza riguardo alla struttura organizzativa;

— la **Componente Linguistica,** che gestisce le conoscenze necessarie per l'interpretazione del contenuto proposizionale delle conversazioni.

Questo articolo focalizza la sua attenzione su questi ultimi due moduli.

2. MODELLI D'UFFICIO: STRUTTURA DI RESPONSABILITÀ E DISTRIBUZIONE DI ESPERIENZE

La necessità di individuare all'interno delle organizzazioni precise competenze professionali è finalizzata ad individuare gli interlocutori che meglio di altri possono fare fronte alle necessità che normalmente si presentano nel lavoro d'ufficio. Queste esigenze possono essere di diversa natura: richieste ed offerte di collaborazione, richieste di distribuzione di informazioni, autorizzazioni per intraprendere attività specifiche. Ad esempio, all'apertura di una conversazione per chiedere l'acquisto di un personal computer (una conversazione aperta da una richiesta di collaborazione), si deve sapere chi interpellare per ricevere l'autorizzazione necessaria allo stanziamento (es. il responsabile del Servizio Acquisti) e chi interpellare per l'acquisto (es. la persona incaricata degli acquisti di PC) e l'installazione. Allo stesso modo, per sapere a chi rivolgersi per la manutenzione: se non esiste in azienda "l'addetto

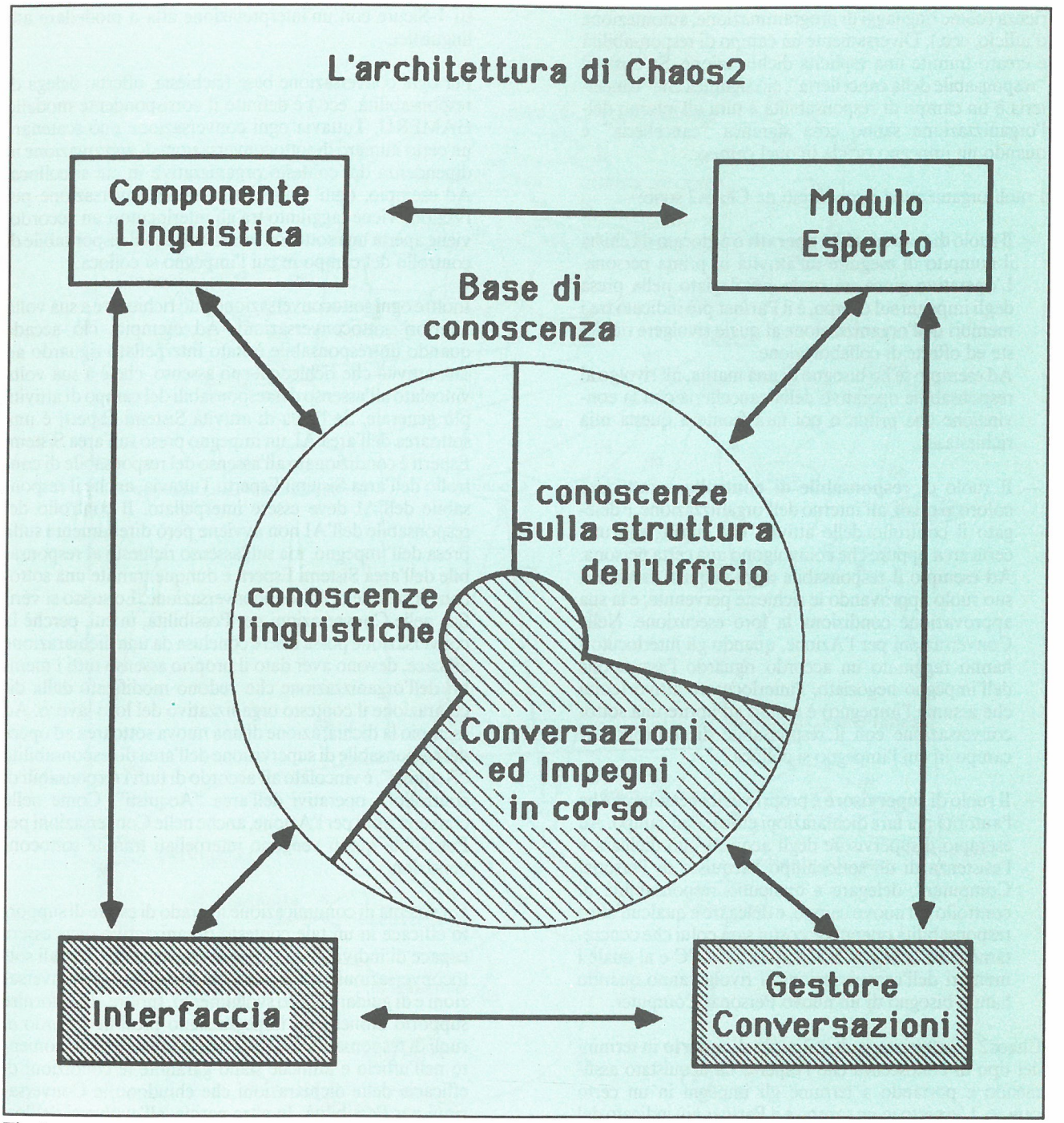


Fig. 1

alla manutenzione" bisognerà ricorrere a quei membri dell'organizzazione che hanno un'esperienza tecnica nell'uso dei personal computers.

Negli uffici esiste una caratterizzazione formale dei ruoli ed una caratterizzazione informale. Il ruolo di direttore generale è un ruolo formale nel senso che il direttore generale è nominato dall'organo competente ed i compiti e le responsabilità di chi gioca tale ruolo sono definiti in modo rigoroso.

Diversamente il ruolo di esperto, ad es. di Office Automation, non è un ruolo formalmente definito nel senso che non corrisponde ad una dichiarazione, ma viene riconosciuto all'interno dell'organizzazione a chi ha acquisito nel tempo l'esperienza che qualifica questo ruolo.

Corrispondentemente, anche la natura dei campi di responsabilità e dei campi di esperienza è diverso. I campi di esperienza esistono per il fatto che un qualche impegno è stato portato a termine riguardo ad una certa area di espe-

rienza (come linguaggi di programmazione, automazione d'ufficio, ecc.). Diversamente un campo di responsabilità è creato tramite una esplicita dichiarazione. Se esiste il "responsabile della cancelleria", ciò significa che "cancelleria è un campo di responsabilità e tutti all'interno dell'organizzazione sanno cosa significa "cancelleria" e quando un impegno ricade in quel campo.

I ruoli organizzativi riconosciuti da Chaos2 sono:

— Il ruolo di **responsabile operativo** è giocato da chi ha il compito di eseguire un'attività in prima persona. L'operativo gioca un ruolo privilegiato nella presa degli impegni sul campo, è il Partner più indicato tra i membri dell'organizzazione al quale rivolgere richieste ed offerte di collaborazione.
Ad esempio se ho bisogno di una matita, mi rivolgo al responsabilie operativo della cancelleria con la convinzione che prima o poi farà fronte a questa mia richiesta.

— Il ruolo di **responsabile di controllo** caratterizza coloro ai quali, all'interno dell'organizzazione, è delegato il controllo delle attività che riguardano una certa area oppure che coinvolgono una certa persona. Ad esempio il responsabile degli acquisti esercita il suo ruolo approvando le richieste pervenute, e la sua approvazione condiziona la loro esecuzione. Nelle Conversazioni per l'Azione, quando gli interlocutori hanno raggiunto un accordo riguardo l'assunzione dell'impegno negoziato, l'interlocutore attivo (colui che assume l'impegno) è tenuto ad aprire una sottoconversazione con il responsabile di controllo del campo in cui l'impegno si colloca.

— Il ruolo di **supervisore** è proprio della persona che ha l'autorità per fare dichiarazioni efficaci sul campo. Ad esempio il supervisore degli acquisti può dichiarare l'esistenza di un sottocampo "acquisto di Personal Computer", delegare a qualcuno responsabilità di controllo sul nuovo campo, e delegare a qualcun'altro responsabilità operativa: costui sarà colui che concretamente si occuperà dell'acquisto dei PC e al quale i membri dell'organizzazione si rivolgeranno quando hanno bisogno di un nuovo Personal Computer.

Chaos2 caratterizza inoltre il ruolo di **esperto** in termini del tipo di conoscenza che l'esperto ha acquistato assumendo e portando a termine gli impegni in un certo campo. L'esperto in un campo è il Partner più indicato dal quale ottenere informazioni cancernenti il campo.

3. STRUTTURE CONVERSAZIONALI COMPLESSE

Ogni ruolo, qui sommariamente descritto, è caratterizzato in modo rigoroso dagli spazi di possibilità nelle Conversazioni per l'Azione e per Possibilità.

Ogni conversazione è rappresentata da un modello GAMERU [3], cioè da Reti (di Petri) di Automi sovrapposti 1-Sicure con un'interpretazione atta a modellare atti linguistici.

Per ogni conversazione base (richiesta, offerta, delega di responsabilità, ecc.) è definito il corrispondente modello GAMERU. Tuttavia ogni conversazione può scatenare un certo numero di sottoconversazioni di approvazione in dipendenza del contesto organizzativo in cui si colloca. Ad esempio, ogni volta che in una Conversazione per l'Azione viene raggiunto tra gli interlocutori un accordo, viene aperta una sottoconversazione con il responsabile di controllo del campo in cui l'impegno si colloca.

Inoltre ogni sottoconversazione può richiedere a sua volta ulteriori sottoconversazioni. Ad esempio, ciò accade quando un responsabile è stato interpellato riguardo ad una attività che richiede il suo assenso, che è a sua volta vincolato all'assenso dei responsabili del campo di attività più generale. Se l'area di attività Sistemi Esperti è una sottoarea dell'area AI, un impegno preso sull'area Sistemi Esperti è condizionato all'assenso del responsabile di controllo dell'area Sistemi Esperti. Tuttavia, anche il responsabile dell'AI deve essere interpellato. Il controllo del responsabile dell'AI non avviene però direttamente sulla presa dell'impegno, ma sull'assenso richiesto al responsabile dell'area Sistemi Esperti e dunque tramite una sottoconversazione della sottoconversazione. Lo stesso si verifica nelle Conversazioni per Possibilità, in cui, perchè la conversazione possa essere conclusa da una dichiarazione efficace, devono aver dato il proprio assenso tutti i membri dell'organizzazione che vedono modificato dalla dichiarazione il contesto organizzativo del loro lavoro. Ad esempio la dichiarazione di una nuova sottoarea ad opera del responsabile di supervisione dell'area di responsabilità "Acquisti", è vincolato all'accordo di tutti i responsabili di controllo e operativi dell'area "Acquisti". Come nelle conversazioni per l'Azione, anche nelle Conversazioni per Possibilità questi vengono interpellati tramite sottoconversazioni.

Un sistema di comunicazione in grado di essere di supporto efficace in un tale contesto organizzativo deve essere capace di individuare, per ogni conversazione, quali sottoconversazioni aprire e con chi, di aprire tali conversazioni e di guidare il loro svolgimento. Inoltre, deve fornire supporto affinché gli impegni siano presi in accordo ai ruoli di responsabilità ed esperienza attivi in ogni momento nell'ufficio e affinché siano garantite le condizioni di efficacia delle dichiarazioni che chiudono le Conversazioni per Possibilità. In altre parole, all'evolversi dell'organizzazione (a seguito delle dichiarazioni e dell'assunzione/assolvimento degli impegni), la struttura dell'organizzazione stessa può non essere nota ai vari membri, il che rende loro difficile individuare di volta in volta il collega più esperto in un certo campo, quello che può autorizzare la prese di una certa commessa, quello che non rifiuterà la presa di una certa commessa, quello che non rifiuterà la presa di un impegno richiestogli, ecc.

Deve essere dunque il sistema che, acquisendo dalla gestione delle conversazioni conoscenza sull'evoluzione dell'organizzazione, fornisce supporto alla risoluzione di

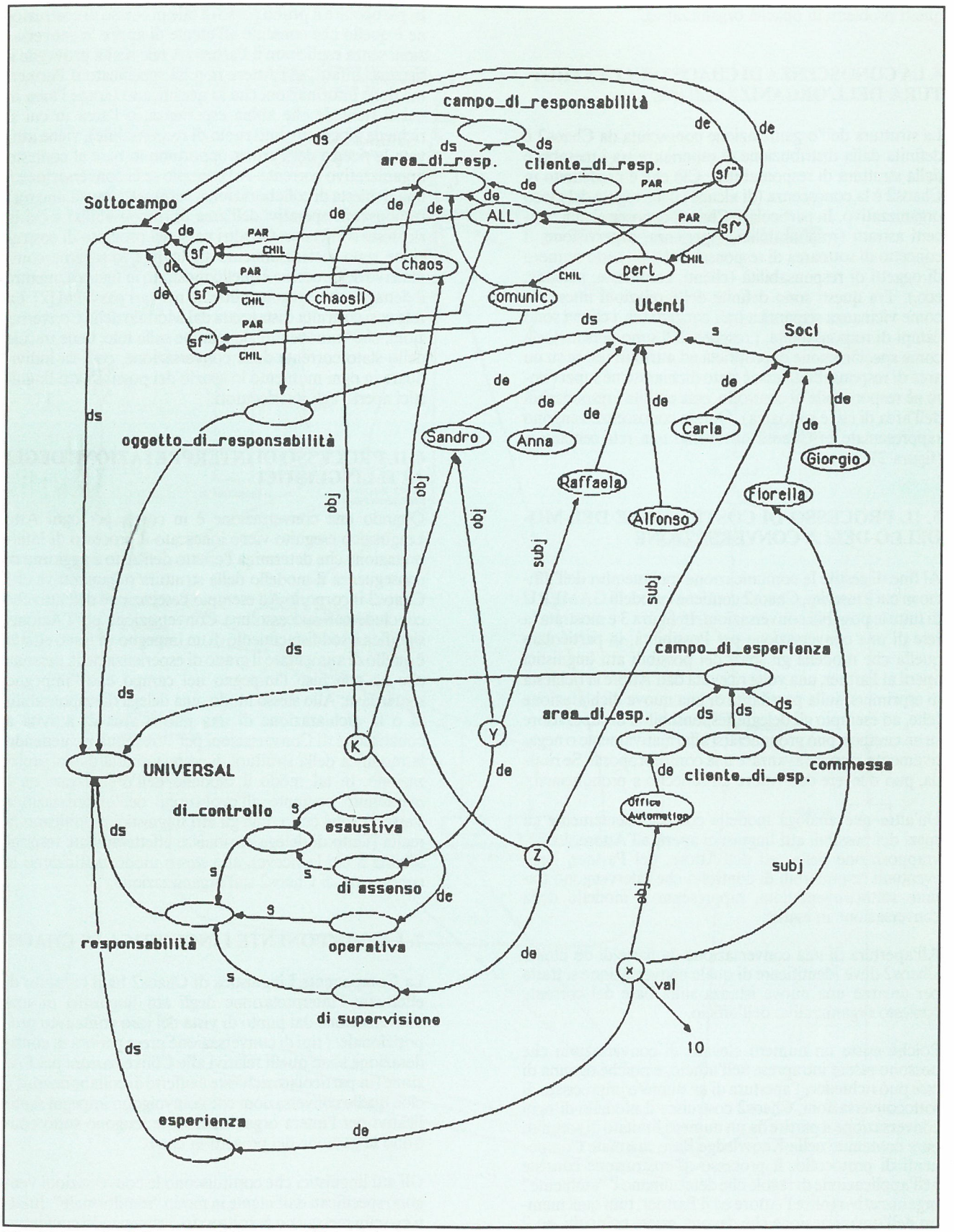


Fig. 2 Esempio di Struttura di Responsabilità e Distribuzione di Esperienze: rete semantica istanziata.

questi problemi di opacità organizzativa.

4. LA CONOSCENZA DI CHAOS2 SULLA STRUTTURA DELL'ORGANIZZAZIONE

La struttura dell'organizzazione conosciuta da Chaos2 è definita dalla distribuzione di esperienze tra i membri e dalla struttura di responsabilità. Ciò che è predefinito in Chaos2 è la conoscenza (di alcune) delle regole del gioco organizzativo. In particolare Chaos2 conosce alcuni concetti astratti (responsabilità, esperienza, supervisione, il concetto di sottoarea di responsabilità), un certo numero di oggetti di responsabilità (clienti, commesse, persone, ecc.). Tra questi sono definite delle relazioni intese sia come vicinanza semantica tra i concetti (es. i clienti sono campi di responsabilità, i responsabili sono persone), sia come specificazione di proprietà ed attributi (es. se su un area di responsabilità non è stato dichiarato né supervisore né responsabile di controllo, essa eredita i responsabili dell'area di cui è sottoarea). Queste conoscenze vengono rappresentate in Chaos2 attraverso una rete semantica (figura 2).

5. IL PROCESSO DI COSTRUZIONE DEL MODELLO DELLA CONVERSAZIONE

Al fine di gestire la comunicazione tra i membri dell'ufficio in cui è inserito, Chaos2 contiene i modelli GAMERU di tutte le possibili conversazioni. In figura 3 è mostrata la rete di una conversazione per Possibilità, in particolare quella che modella gli spazi dei possibili atti linguistici aperti al Partner, una volta ricevuta dall'Attore la richiesta di esprimersi sulla possibilità di una nuova dichiarazione (che, ad esempio gli deleghi responsabilità di supervisore su un campo), può pronunciarsi affermativamente o negativamente, oppure avanzare una controproposta. Se ritarda, può ricevere dall'Attore un sollecito a pronunciarsi.

Un'altra rete analoga modella corrispondentemente gli spazi dei possibili atti linguistici aperti all'Attore. La sovrapposizione delle reti dell'Attore, del Partner, degli eventuali responsabili di controllo che intervengono tramite sottoconversazioni, rappresenta il modello della conversazione in esame.

All'apertura di una conversazione da parte di un utente Chaos2 deve identificare di quale conversazione si tratta per crearne una nuova istanza sulla base del corrente contesto organizzativo dell'ufficio.

Poiché esiste un numero elevato di conversazioni che possono essere intraprese nell'ufficio, e poiché ognuna di esse può richiedere l'apertura di un numero imprecisato di sottoconversazioni, Chaos2 costruisce il modello di ogni conversazione a partire da un numero limitato di sottoreti base contenute nella Knowledge Base chiamate Componenti di protocollo. Il processo di costruzione consiste nell'applicazione di regole che determinano l'"ambiente" organizzativo (oltre l'Attore ed il Partner, tutti quei membri dell'organizzazione che devono essere coinvolti, ed il ruolo giocato da ciascuno nella conversazione).

In particolare il primo passo di tale processo di costruzione è quello che consente all'utente di aprire le conversazioni senza esplicitare il Partner. A tale scelta provvede il sistema. Infatti, se l'Attore non ha specificato il Partner, ma delle informazioni che lo qualificano (come l'area in cui si richiede che abbia esperienza, o l'area in cui si richiede giochi un dato ruolo di responsabile), viene attivata la ricerca del Partner opportuno in base al contesto organizzativo corrente. Ad esempio se la conversazione è una richiesta di collaborazione, verrà selezionata uno tra i responsabili operativi dell'area di responsabilità a cui la richiesta si riferisce. Gli altri passi del processo di costruzione sono sostanzialmente analoghi e lo schema complessivo del processo è quello mostrato in figura 4, mentre il dettaglio delle regole utilizzate nei vari passi è in [2]. La rete così costruita è istanziata dal Modulo delle Conversazioni, che "muovendo" le marche sulla rete, tiene traccia dello stato corrente della conversazione, così da individuare in ogni momento lo spazio dei possibili atti linguistici aperto agli interlocutori.

6. IL PROCESSO DI INTERPRETAZIONE DEGLI ATTI LINGUISTICI

Quando una conversazione è in corso, per ogni Atto Linguistico eseguito viene innescato il processo di interpretazione che determina l'effetto dell'Atto e aggiorna di conseguenza il modello della struttura organizzativa che Chaos2 incorpora. Ad esempio l'esecuzione dell'atto che conclude con successo una Conversazione per l'Azione, significa il soddisfacimento di un impegno ed il suo effetto è quello di aumentare il grado di esperienza della persona che ha concluso l'impegno nel campo che l'impegno riguardava. Allo stesso modo, una delega di responsabilità o la dichiarazione di una nuova area di attività a conclusione di Conversazioni per Possibilità richiedendo la modifica della struttura di responsabilità dell'organizzazione. In tal modo il modello dell'organizzazione è mantenuto coerente all'evoluzione dell'organizzazione stessa, poiché così come gli atti linguistici modificano la realtà (l'atto di delega attribuisce effettivamente responsabilità a chi la riceve), allo stesso modo modificano le conoscenze di Chaos2 sull'organizzazione.

7. LA COMPONENTE LINGUISTICA DI CHAOS

La Componente Linguistica di Chaos2 ha il compito di effettuare l'interpretazione degli atti linguistici di una conversazione dal punto di vista del loro contenuto proposizionale. I tipi di conversazione prese per ora in considerazione sono quelli relativi alle Conversazioni per l'Azione (in particolare richieste e offerte di collaborazione), cioè quelle conversazioni che coinvolgono impegni significativi per l'intera organizzazione e tengono sotto controllo la gestione dei progetti in corso.

Gli atti linguistici che costituiscono le conversazioni vengono specificati dall'utente in modo "semiformale". Infatti ciascun messaggio è strutturato distinguendo contenuto illocutivo, che viene dichiarato formalmente, e contenuto

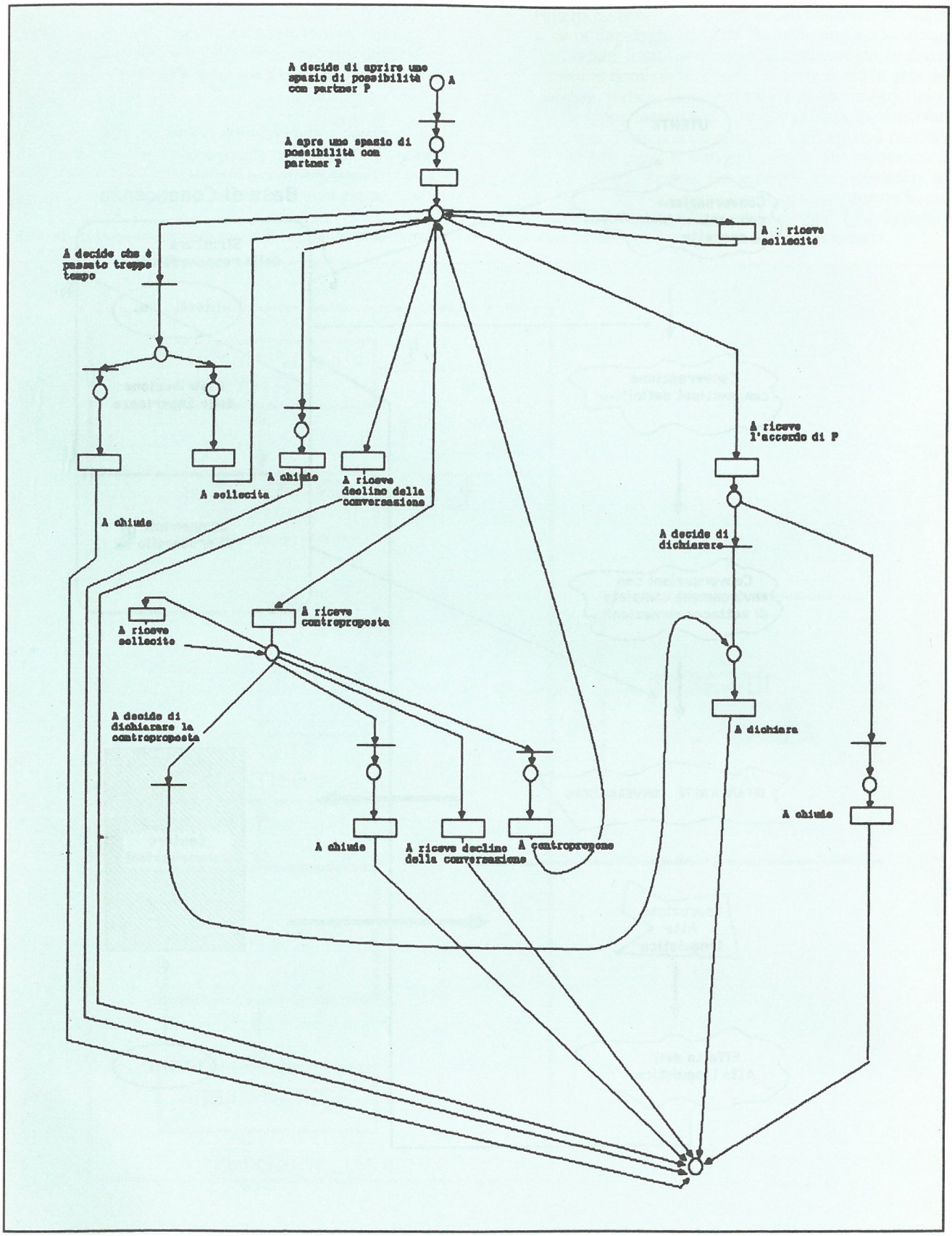


Fig. 3 Conversazione per Possibilità.

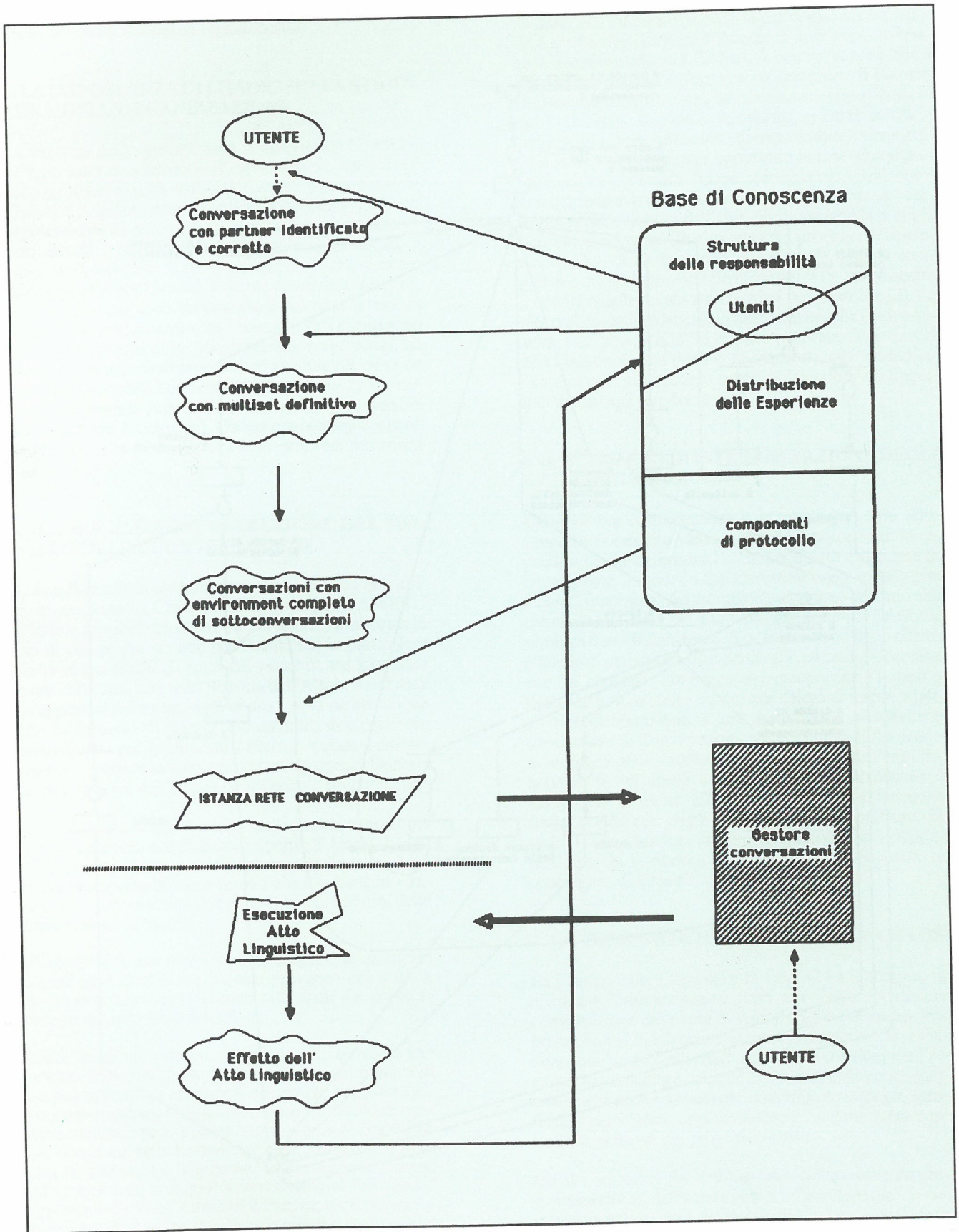


Fig. 3 Processo di costruzione del modello SA della conversazione.

proposizionale, che, invece, è specificato utilizzando il linguaggio naturale. Quindi, dal punto di vista del contenuto proposizionale, le conversazioni sono viste come una sequenza di espressioni, ciascuna delle quali descrive un'azione.

In Chaos2, interpretare il contenuto proposizionale di un atto linguistico (battuta) significa capire qual'è l'azione richiesta o offerta (controfferta o controrichiesta) da uno dei due interlocutori nella conversazione in esame.

A questo scopo, assume un ruolo di primaria importanza l'interpretazione dei sintagmi verbali presenti nelle espressioni in linguaggio naturale che compongono la battuta [11]. Infatti, fissato un dominio di applicazione, ne deriva il relativo dominio lessicale i cui verbi e, più in generale sintagmi verbali, sono denotatori di particolari tipi di azione (appartenenti al dominio in esame). In particolare sulla base di un preciso dominio applicativo è possibile identificare classi si sintagmi verbali che esprimono lo stesso tipo di Azione. Per esempio, se si considera un ambiente di lavoro come quello di una software house, l'azione "strutturare il sistema in moduli" è equivalente a "fare la scomposizione in moduli del sistema".

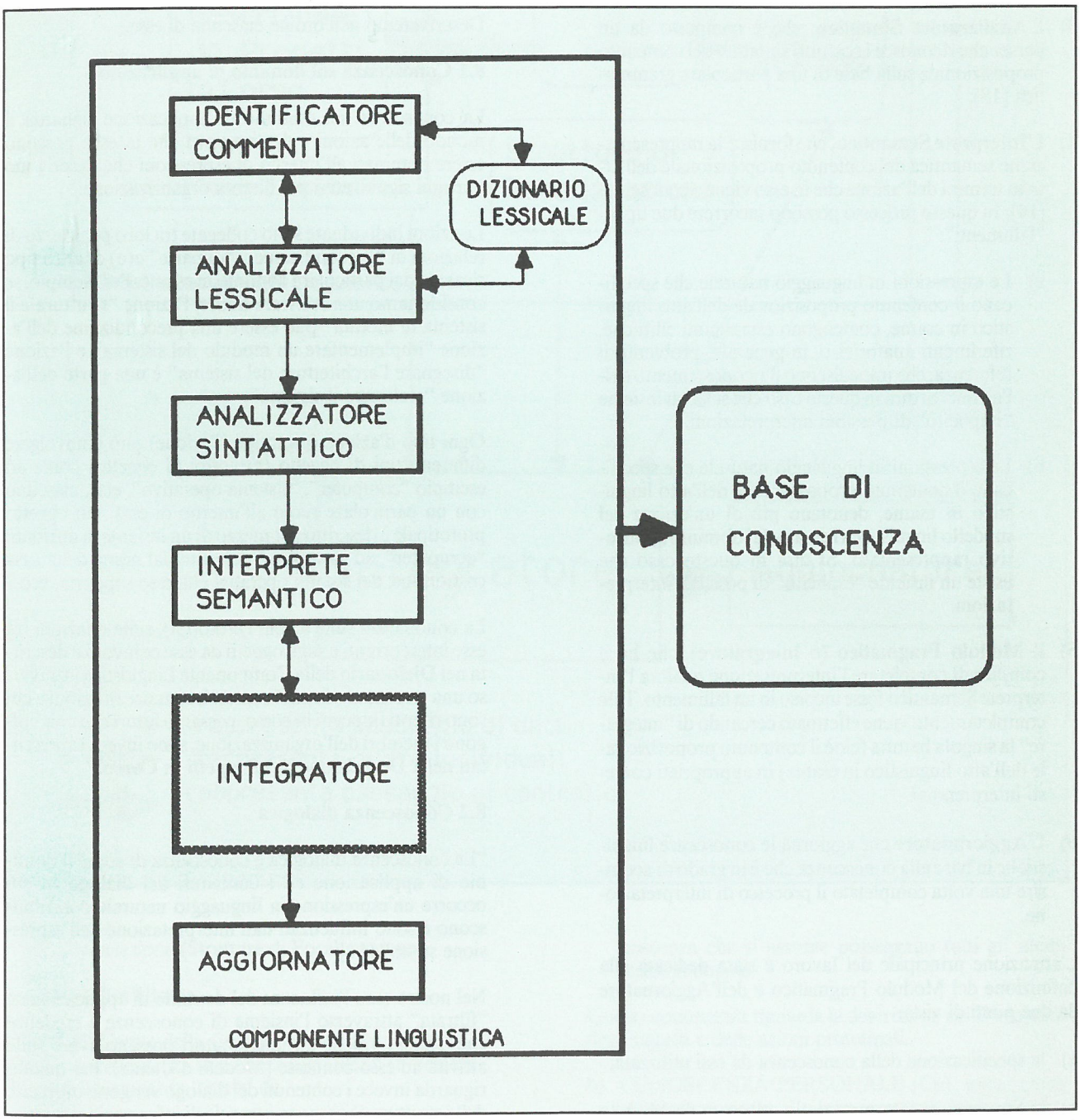


Fig. 5 La Componente Linguistica di Chaos2.

La Componente Linguistica può essere scomposta in una serie di blocchi che identificano passi distinti del processo interpretativo (vedi figura 5):

1) L'**Identificatore Commenti,** che scarta dal contenuto proposizionale tutto ciò che riconosce essere commento e quindi non direttamente significativo per la descrizione dell'azione;

2) L'**Analizzatore Lessicale,** che esegue il riconoscimento delle singole parole del contenuto proposizionale dell'atto linguistico;

3) L'**Analizzatore Sintattico,** che è composto da un parser che riconosce i costrutti sintattici del contenuto proposizionale sulla base di una particolare grammatica [18];

4) L'**Interprete Semantico,** che fornisce la rappresentazione semantica del contenuto proposizionale dell'atto in termini dell'azione che in esso viene riconosciuta [14]. In questo processo possono incorrere due tipi di "fallimenti":

   a) Le espressioni in linguaggio naturale che specificano il contenuto proposizionale dell'atto linguistico in esame, contengono espressioni ellittiche, riferimenti anaforici o, in generale, problemi di referenza, che impediscono il riconoscimento dell'azione. Si dirà in questo caso che si ha un insieme "implicito" di possibili interpretazioni.

   b) Le espressioni in linguaggio naturale che specificano il contenuto proposizionale dell'atto linguistico in esame, denotano più di un'azione nel modello linguistico ricavato dal dominio applicativo rappresentato. Si dirà in questo caso che esiste un insieme "esplicito" di possibili interpretazioni.

5) Il **Modulo Pragmatico** (o **Integratore**), che ha il compito di completare l'interpretazione qualora l'Interprete Semantico fosse incorso in un fallimento. Tale completamento viene effettuato cercando di "integrare" la singola battuta (cioè il contenuto proposizionale dell'atto linguistico in esame) in appropriati contesti interpretativi.

6) L'**Aggioranatore** che aggiorna le conoscenze linguistiche in base alla conoscenza che è in grado di acquisire una volta completato il processo di interpretazione.

L'attenzione principale del lavoro è stata dedicata alla definizione del Modulo Pragmatico e dell'Aggiornatore da due punti di vista:

a) la specificazione della conoscenza da essi utilizzata;

b) le Strategie di Interpretazione utilizzate da Modulo Pragmatico.

8. LA CONOSCENZA NECESSARIA ALLA COMPONENTE LINGUISTICA

La conoscenza utilizzata dalla Componente Linguistica di Chaos2 e, in particolare, dall'Integratore è suddivisa in due tipi di conoscenza (vedi figura 6):

a) Conoscenza sul dominio di applicazione: **Dizionario e Descrizione dei progetti in corso;**

b) Conoscenza dialogica: **Modelli d'Utente** e **Strutture di Focalizzazione.**

Descriveremo nell'ordine ciascuna di esse.

8.1 Conoscenza sul dominio di applicazione

La conoscenza sul dominio di applicazione riguarda il mondo delle azioni e degli oggetti che in esso possono essere nominati all'interno di espressioni che descrivono impegni significativi per l'intera organizzazione.

Le azioni individuate sono collegate tra loro per mezzo di relazioni di "precondizione", "di parte" o(e) di altro tipo ricavate dal particolare dominio in esame. Per esempio, se consideriamo una software house, l'azione "strutturale il sistema in moduli" può essere una precondizione dell'azione "implementare un modulo del sistema" e l'azione "disegnare l'architettura del sistema" è una parte dell'azione "realizzare il sistema".

Ogni tipo d'azione (prototipo d'azione) può coinvolgere differenti tipi di oggetto (prototipi di oggetto) come ad esempio "computer", "sistema operativo", ecc., ciascuno con un particolare ruolo all'interno di essa. Un oggetto prototipale è descritto per mezzo di un insieme di attributi: "computer", ad esempio, è descritto dal nome della casa costruttrice, dai sistemi operativi che esso supporta, ecc..

La conoscenza sulle azioni (prototipi), sulle relazioni tra esse intercorrenti e sugli oggetti da esse coinvolti è descritta nel **Dizionario** della Componente Linguistica attraverso una rete semantica di Frames. Le istanze di azioni a cui sono riferiti impegni in corso, passati o futuri che coinvolgono i membri dell'organizzazione, sono invece rapresentati nella **Descrizione dei Progetti in Corso.**

8.2 Conoscenza dialogica

"La conoscenza dialogica è conoscenza di come il dominio di applicazione ed i contenuti del dialogo in cui occorre un'espressione in linguaggio naturale contribuiscono e sono influenzati dall'interpretazione dell'espressione stessa" [11].

Nel nostro caso l'influenza del dominio di applicazione è "filtrata" attraverso l'insieme di conoscenze e credenze che Chaos2 ritiene gli interlocutori possano avere sulle attività ad esso connesse (Modelli d'Utente). Per quanto riguarda invece i contenuti del dialogo vengono utilizzate delle strutture che tengono traccia di informazioni semantiche e sintattiche associate alle battute che si susseguono

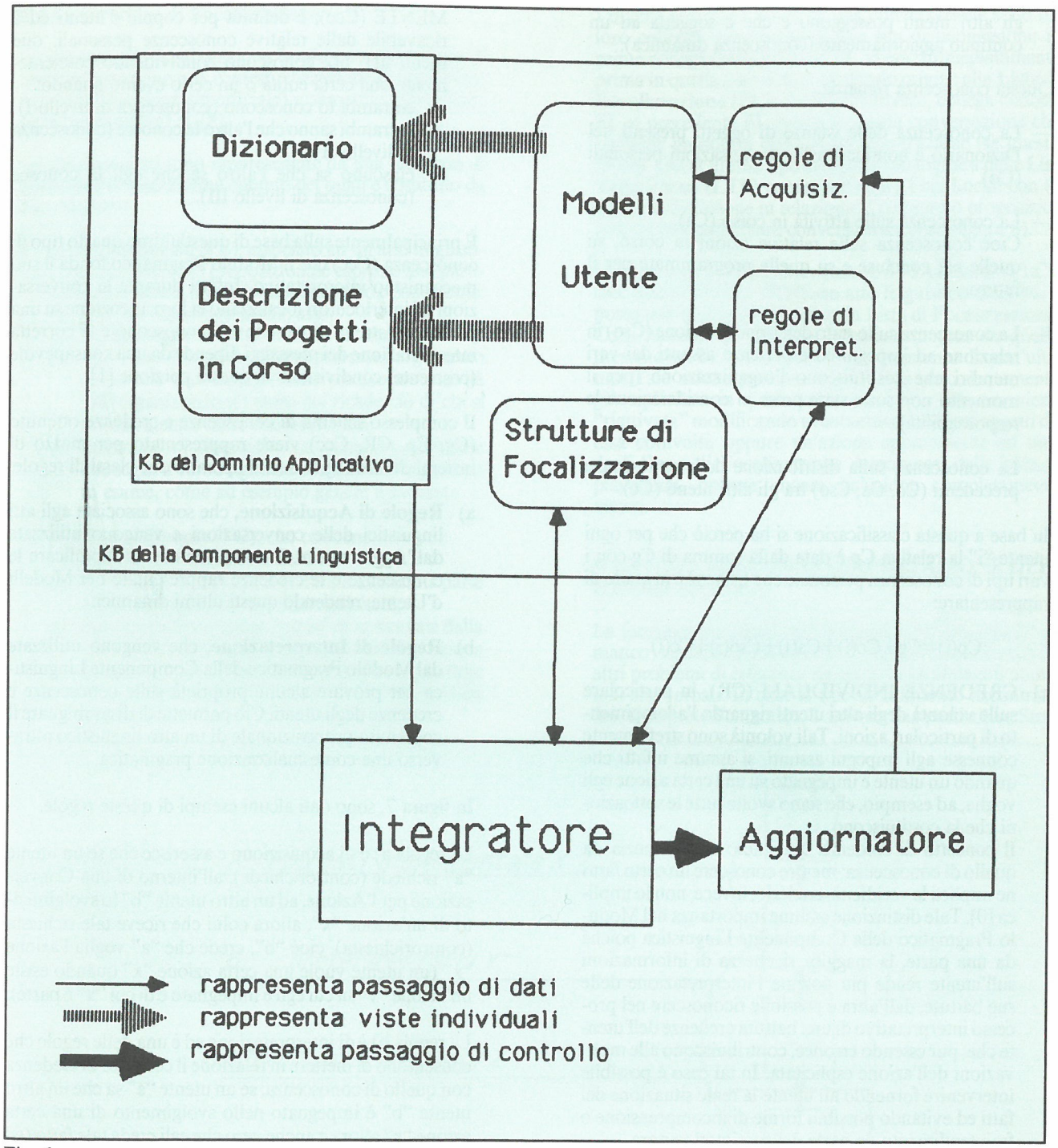


Fig. 6.

in una conversazione (Strutture di Focalizzazione).

8.2 I Modelli d'Utente

I Modelli d'Utente della Componente Linguistica costituiscono la conoscenza di Chaos2 sugli utenti (i membri dell'organizzazione) in termini di:

a) CONOSCENZA GLOBALE (Cg), inteso come conoscenza che si assume posseggano tutti gli utenti indistintamente.

Questa conoscenza riguarda la descrizione nel Dizionario degli oggetti e delle azioni prototipali.

b) CONOSCENZA PERSONALE (Cp), inteso come conoscenza legata alle esperienze personali (conversazioni personali), che quindi non é detto che anche

gli altri utenti posseggono e che è soggetta ad un continuo aggiornamento (conoscenza dinamica).

Questa conoscenza riguarda:

— La conoscenza delle istanze di oggetti presenti nel Dizionario e nominati nelle conversazioni personali (Co).

— La conoscenza sulle attività in corso (Ca).
Cioè conoscenza sulle relative azioni in corso, su quelle già concluse e su quelle programmate per il futuro.

— La conoscenza sullo stato dell'organizzazione (Cso) in relazione ad impegni ed esperienze assunti dai vari membri che costituiscono l'organizzazione (per il momento non sono state prese in considerazione le responsabilità).

— La conoscenza sulla distribuzione delle conoscenze precedenti (Co, Ca, Cso) tra gli altri utenti (Cc).

In base a questa classificazione si ha perciò che per ogni utente "i" la relativa Cp è data dalla somma di Cg con i vari tipi di conoscenza personale che Chaos2 è in grado di rappresentare:

$$Cp(i)=Cg+Co(i)+Ca(i)+Cso(i)+Cc(i)$$

c) CREDENZE INDIVIDUALI (CR), in particolare sulle volontà degli altri utenti riguardo l'adempimento di particolari azioni. Tali volontà sono strettamente connesse agli impegni assunti, si assume infatti che quando un utente è impegnato su una certa azione egli voglia, ad esempio, che siano svolte tutte le sottoazioni che la costituiscono.
Il concetto di credenza qui inteso si differenzia da quello di conoscenza: mentre conoscere un certo fatto ne implica la veridicità, crederlo, invece, non lo implica [0]. Tale distinzione assume importanza nel Modulo Pragmatico della Componente Linguistica poiché da una parte, la maggior ricchezza di informazioni sull'utente rende più potente l'interpretazione delle sue battute, dall'altra è possibile riconoscere nel processo interpretativo di una battuta credenze dell'utente che, pur essendo erronee, contribuiscono alle motivazioni dell'azione esplicitata. In tal caso è possibile intervenire fornendo all'utente la reale situazione dei fatti ed evitando possibili forme di incomprensione o fraintendimento da parte del suo interlocutore.

d) CONOSCENZA CONDIVISA COSCIENTEMENTE (Ccc), è definita per coppie d'utenti ed è ricavabile dalle relative conoscenze personali: due utenti ut1, ut2 conoscono condividendo coscientemente una certa entità o un certo evento quando:
— entrambi lo conoscono (conoscenza di livello I)
— entrambi sanno che l'altro la conosce (conoscenza di livello II)
— ciascuno sa che l'altro sa che egli la conosce (conoscenza di livello III).

È principalmente sulla base di quest'ultimo quarto tipo di conoscenza (Ccc) che il Modulo Pragmatico fonda il suo meccanismo interpretativo. Infatti durante le conversazioni gli interlocutori focalizzano la loro attenzione su una porzione ristretta di ciò che essi conoscono e la corretta interpretazione dei messaggi dipende da una consapevole (cosciente) condivisione di questa porzione [1].

Il complesso schema di conoscenze e credenze ottenuto (Cg, Cp, CR, Ccc) viene rappresentato per mezzo di insiemi di fatti logici, su cui operano due classi di regole:

a) **Regole di Acquisizione,** che sono associate agli atti linguistici delle conversazioni e vengono utilizzate dall'Aggiornatore. Esse consentono di modificare le conoscenze e le credenze rappresentate nei Modelli d'Utente, rendendo questi ultimi dinamici.

b) **Regole di Interpretazione,** che vengono utilizzate dal Modulo Pragmatico della Componente Linguistica per provare alcune proprietà sulle conoscenze e credenze degli utenti. Ciò permette di disambiguare il contenuto proposizionale di un atto linguistico attraverso una contestualizzazione pragmatica.

In figura 7, sono dati alcuni esempi di queste regole.

La regola a) è di acquisiziono e asserisce che se un utente "a" richiede (controrichiede), all'interno di una Conversazione per l'Azione, ad un altro utente "b" lo svolgimento di un'azione "x", allora colui che riceve tale richiesta (controrichiesta), cioè "b", crede che "a" voglia l'azione "x" (un utente vuole una certa azione "x" quando esiste un'azione "y" su cui egli è impegnato e di cui "x" è parte).

La regola b) è di interpretazione ed è una delle regole che consentono di mettere in relazione il concetto di credenza con quello di conoscenza: se un utente "a" sa che un altro utente "b" è impegnato nello svolgimento di una certa azione "x" allora è anche vero che egli crede tale fatto (ed inoltre che tale fatto è di persé vero in quel particolare istante nella base di conoscenza di Chaos2).

$$a)\ \forall a,b,x\ (\mathbf{REQUEST}(a,b,x) \Rightarrow \mathbf{CRv}(b,a,x))$$

$$b)\ \forall a,b,x\ (\mathbf{Ci}(a,b,x) \Rightarrow \mathbf{CRi}(a,b,x) \wedge \mathbf{E'\text{-}IMPEGNATO}(b,x))$$

Fig. 7

8.4 Le Strutture di Focalizzazione

Le Strutture di Focalizzazione forniscono alle battute di una conversazione una contestualizzazione di tipo sintattico (focalizzazione superficiale) e semantico (focalizzazione profonda) [13], [8].

Le conversazioni sono rappresentate da una sequenza di **Spazi di Focalizzazione** ognuno dei quali è costituito da due strutture:

— un **Focus Locale** (FL), associato ad un atto linguistico il cui contenuto proposizionale esprime una particolare azione del dominio rappresentato e contenente:
  a) l'azione esplicitata in tale atto,
  b) un indicatore di colui che l'ha formulata,
  c) un **Focus Tematico,** che rappresenta l'argomento principale (ad esempio una particolare attività dell'organizzazione) entro cui ricade ciò di cui si sta discutendo nella conversazione,
  d) informazioni sintattiche sul sintagma nominale che segue il verbo significativo dell'atto linguistico in esame, come ad esempio genere e numero.

— Un **Focus Globale** (FG), che contiene:
  a) l'azione (az) contenuta nel FL,
  b) l'azione (se esiste) di cui az è stata dichiarata parte nell'atto linguistico che l'ha esplicitata,
  c) tutte le possibile azioni "parte" di az ricavate dalla conoscenza Ccc dei due interlocutori. Se l'azione del punto b) non esiste, il FG contiene anche tutte le possibili azioni di cui az "è parte" ricavate sempre dalla Ccc dei due interlocutori.

I diversi foci locali creati in una conversazione sono tra loro collegati per mezzo di due tipi di connessioni: la prima collega ciascun FL all'FL creato immediatamente prima in quella conversazione dando origine alla **Lista di Focalizzazione (FL);** il secondo, invece, collega ciascun FL al precedente FL creato in quella conversazione che contiene un'istanza dello stesso tipo d'azione (se questo esiste). Quest'ultimo tipo di legame da origine a delle **List Type Focus (LTF)** contenenti tutti i Foci Locali con lo stesso tipo d'azione in relazione al contenuto proposizionale di una particolare conversazione (vedi figura 8).

In ogni istante della conversazione il FL corrente (FLC), cioè quello relativo all'ultimo atto linguistico eseguito, è posto nell'ultima posizione della Lista di Focalizzazione. Ogni battuta che esprime un'azione diversa da quella presente nel FLC determina il cambiamento di quet'ultimo. Tale azione può essere un'azione precedentemente già esplicitata in quella conversazione e che ora viene "riattivata" modificando eventualmente alcuni oggetti da essa coinvolti, oppure un'azione appartenente ad uno degli Spazi di Focalizzazione determinati dalle battute precedenti, oppure ancora un'azione completamente nuova.

9. LE STRATEGIE DI INTERPRETAZIONE DELL'INTEGRATORE

La funzione principale dell'Integratore (il Modulo Pragmatico) riguarda la risoluzione di riferimenti anaforici e di altri problemi di referenza riguardanti sia sintagmi nominali che verbali. In base al tipo di fallimento riscontrato dall'Interprete Semantico, è attivata una particolare Stra-

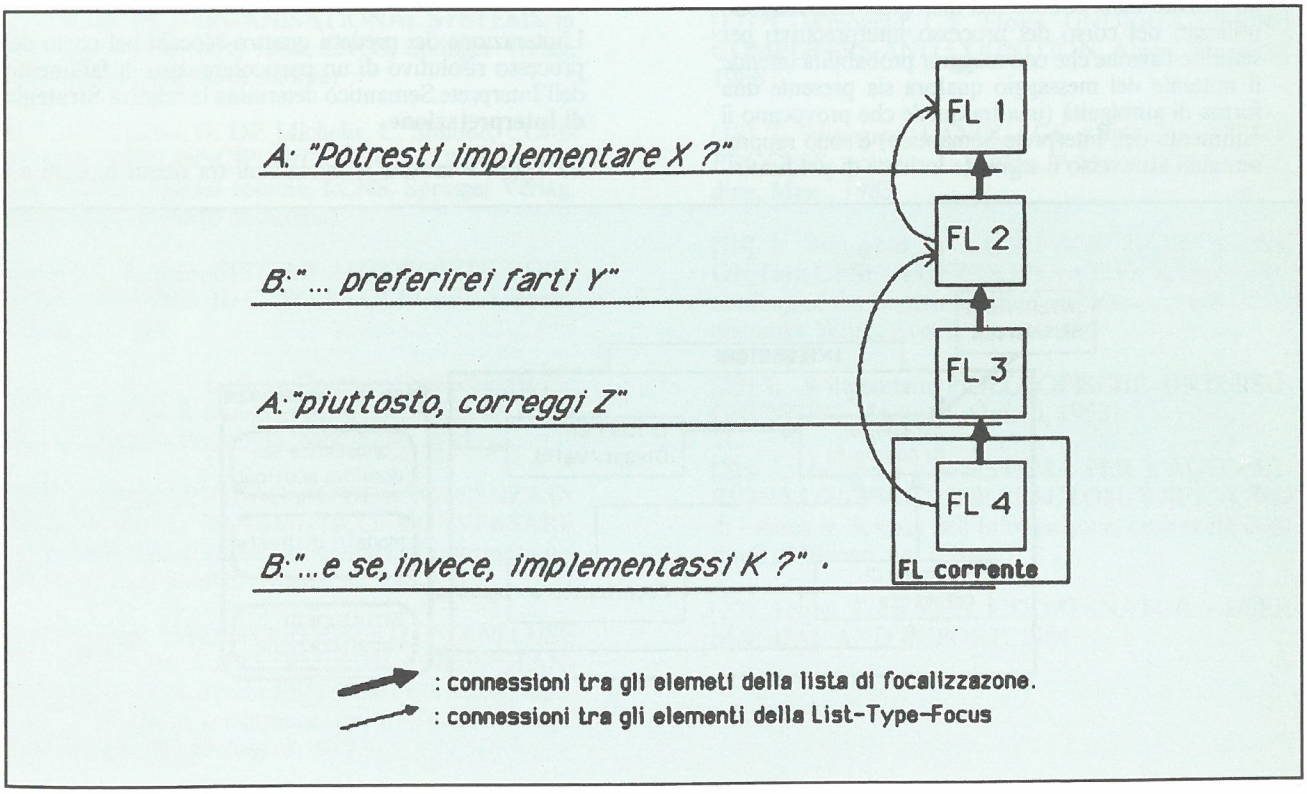


Fig. 8

tegia di Interpretazione che utilizza le fonti di conoscenza descritte precedentemente per compiere i propri processi decisionali. In particolare, per quanto riguarda i Modelli d'Utente, assume una significativa importanza la distinzione tra Cp e Ccc, in quanto si assume che, per la corretta interpretazione del contenuto proposizionale di un atto linguistico, la porzione di conoscenza su cui è focalizzata l'attenzione degli interlocutori debba essere condivisa coscientemente. In caso contrario, cioè ad esempio essa appartiene solamente alla conoscenza personale di uno dei due, Chaos2 è ugualmente in grado di effettuare l'interpretazione ma, oltre a ciò, è anche in grado di segnalare una possibile incomprensione da parte di chi riceverà il messaggio in esame. Tutto ciò consente quindi di eliminare alcune possibili forme di incomprensione o di fraintendimento tra i partecipanti alla conversazione.

La struttura logico-funzionale dell'Integratore è costituita da quattro "blocchi" (fig. 9):

1) Il **Solutore di Referenza,** che permette di risolvere i riferimenti anaforici e le espressioni ellittiche presenti nella battuta utilizzando informazioni fornite dal Focalizzatore, dal Dizionario e dai Modelli d'Utente (in particolare Co e Ccco, la conoscenza condivisa coscientemente sugli oggetti, che restringono l'insieme dei possibili oggetti denotati a quelli conosciuti dall'utente).

2) Il **Focalizzatore,** che estrae informazioni dagli Spazi di Focalizzazione, crea nuovi Foci locali e percorre la Lista di Focalizzazione ricercando il nuovo Focus Locale Corrente o la nuova List-Type-Focus da attivare.

3) I **Criteri di Disambiguazione/Validazione,** che sono ricavati dalla conoscenza dialogica. Essi vengono utilizzati nel corso del processo interpretativo per stabilire l'azione che con maggior probabilità intende il mittente del messaggio qualora sia presente una forma di ambiguità (una tra quelle che provocano il fallimento dell'Interprete Semantico) e sono rappresentabili attraverso il seguente insieme di predicati:

   ↕ ACTION-SEMANTIC-CHECK
   ↕ FOCALIZATION-CHECK
   ↕ COOPERATION-CHECK
   ↕ WANT-CHECK
   ↕ IMPROBABILITY-CHECK
   ↕ POSSIBLE-RELATION-CHECK

   Ad esempio il "COOPERATION-CHECK" si basa sull'ipotesi di cooperatività tra i membri dell'organizzazione. Esso si applica solamente se l'atto linguistico cui appartiene la battuta in esame è un'offerta o una controfferta. Una particolare azione soddisfa il "COOPERATION-CHECK" se, nel modello d'utente relativo a colui che ha eseguito l'atto linguistico, è presente la credenza (CR) circa il fatto che l'attuale interlocutore voglia l'azione in questione.

   Il "WANT-CHECK", invece, si applica solamente se l'atto linguistico cui appartiene la battuta è una richiesta o una controrichiesta. Una particolare azione soddisfa il "WANT-CHECK" se, dal modello d'utente relativo a colui che ha eseguito l'atto linguistico in esame, è possibile inferire che egli vuole tale azione.

4) Le **Strategie di Controllo,** che determinano:
   a) come ricavare la successione di azioni candidate alla risoluzione interagendo con il Solutore di Referenze e il Focalizzatore (caso in cui si ha un insieme di possibili interpretazioni implicito).
   b) Quali Criteri di Disambiguazione/Validazione tra quelli previsti dal modello linguistico applicare e secondo quale ordine.
   c) Come risalire gli spazi di focalizzazione determinati dal contenuto proposizionale della conversazione nel corso del processo interpretativo.

L'interazione dei predetti quattro blocchi nel corso del processo risolutivo di un particolare caso di fallimento dell'Interprete Semantico determina la relativa **Strategia di Interpretazione.**

La figura 9 mostra le interazioni tra questi blocchi e i

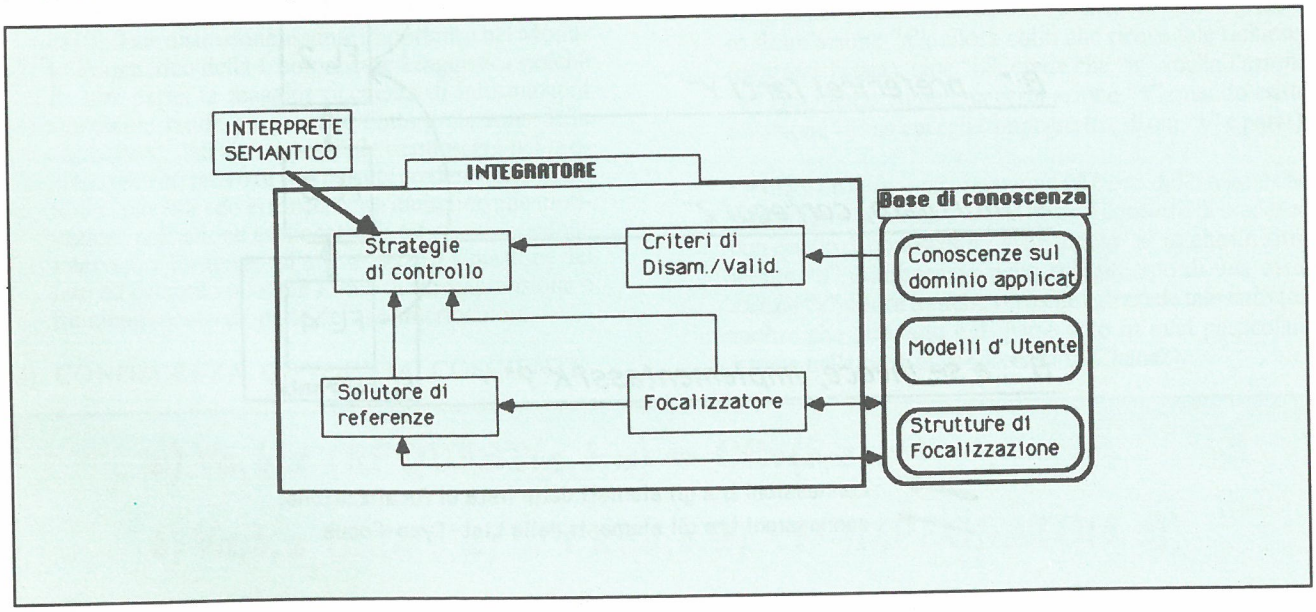


Fig. 9

passaggi di controllo e dati tra Integratore e gli altri moduli della Componente Linguistica.

In [2] sono descritti con maggiore dettaglio i quattro moduli dell'Integratore e viene data una particolareggiata descrizione delle Strategie di Interpretazione.

10. RINGRAZIAMENTI

Si rivolge un particolare ringraziamento a Fiorella De Cindio ed a Carla Simone del Dipartimento di Scienze dell'Informzione dell'Università degli Studi di Milano che hanno seguito con interesse la stesura di questo articolo fornendo preziosi consigli.

Questo lavoro è stato parzialmente finanziato dal Ministro della Pubblica Istruzione.

11. BIBLIOGRAFIA

[1] A. Bonomi, LE VIE DEL RIFERIMENTO, Studi Bompiani 16, 1975

[2] S. Bottarelli, RUOLI ORGANIZZATIVI E SISTEMI DI COMUNICAZIONE: CONVERSARE IN CHAOS, Tesi di Laurea in Scienze dell'Informazione, Università degli Studi di Milano, aa. 1985-1986

[3] F. De Cindio, G. De Michelis, C. Simone, GAMERU: A LANGUAGE FOR THE ANALISYS AND DESIGN OF HUMAN COMMUNICATION PRAGMATICS WHITING ORGANISATIONAL SYSTEMS, in Advances in Petri Nets 1986, LNCS, Springer Verlag, Berlin 1987, (in corso di stampa)

[4] F. De Cindio, G. DE Michelis, C. Simone, COMMUNICATION DISCIPLINES OF CHAOS, Advances on Petri Nets, special colume, LCNS, Springer Verlag, Berlin 1987, (in corso di stampa)

[5] G. De Michelis, SISTEMI AUTOPOIETICI DEL TERZO ORDINE: IL CASO DEGLI UFFICI, Metamorfosi, 3, 1986

[6] C.F. Flores, J.J. Ludlow, Doing and Speaching in the Office, in F. Fish, R. Sprague (eds.), DDS: ISSUES AND CHALLENGES, Pergamon, New York, 1981

[7] A. Gerevini, LINGUAGGIO E CONOSCENZA IN UN CONTESTO PRAGMATICO: CONVERSARE IN CHAOS, Tesi di Laurea in Scienze dell'Informazione, Università degli Studi di Milano, aa. 1985-86

[8] B.J. Grosz, THE INTERPRETATION AND USE OF FOCUS IN A SYSTEM FOR UNDERSTANDING DIALOGS, Proceedings of the Fifth Joint Conference on Artificial Intelligence, pp. 67-76, Cambridge, Massachusetts, 22-25 August 1977

[9] Hintikka, KNOWLEDGE AND BELIEF, Ithaca New York: Cornell University Press 1962

[10] H. Maturana, F. Valera, AUTOPOIESI E COGNIZIONE, LA REALIZZAZIONE DEL VIVENTE, Marsilio Editori, Venezia, 1985

[11] A.E. Robinson, THE INTERPRETATION OF VERB PHRASES IN DIALOGS, Tecnical Note 206, Artificial Intelligence Center, Computer Science Technology Division, SRI Progect 5844

[12] J.R. Searle, SPEECH ACTS: AN ESSAY IN THE PHILOSOPHY OF LANGUAGE, Cambridge University Press, Cambridge, 1969

[13] C.L. Sidner, FOCUSING FOR INTERPRETATION OF PRONOUNS, American Journal of Computational Linguistics, volume 7, numero 4, 1981

[14] H. Tennant, NATURAL LANGUAGE PROCESSING, Petrocelli Books, 1981

[15] R. Vassallo, APPLICAZIONE DEI SISTEMI ESPERTI ALL'AUTOMAZIONE D'UFFICIO: I COORDINATOR DELLA SECONDA GENERAZIONE, Tesi di Laurea in Scienze dell'Informazione, Università degli Studi di Milano, aa. 1984-85

[16] R. Vassallo, A. Zanaboni, APPLICAZIONE DEI SISTEMI ESPERTI NELLA AUTOMAZIONE D'UFFICIO: IL PROTOTIPO CHAOS, Note di Software, marzo 1986

[17] T. Winograd, C.F. Flores, UNDERSTANDING COMPUTERS AND COGNITION, Ablex, Norwood, 1986

[18] T. Winograd, LANGUAGE AS A COGNITIVE PROCESS, volume 1: Syntax, Addison-Wesley, Reading, Mass., 1983

[19] T. Winograd. A LANGUAGE PERSPECTIVE ON THE DESIGN OF COOPERATIVE WORK, Proceedings of the Conference on Computer-Supported Cooperative Work, Austin, Texas, 1986

[20] L. Wittgenstain PHILOSOPISCHE UNTERSUCHUNGEN, Blackwell, Oxford, 1953

[21] A. Zanaboni UN SISTEMA PER L'AUTOMAZIONE D'UFFICIO CHE SI AUTOMODIFICA, Tesi di Laurea in Scienze dell'Informazione, Università degli Studi di Milano, aa. 1984-85

[22] Action Tecnologies, COORDINATOR - USER MANUAL AND REPORT, 1984

ADVANCED PROGRAM GENERATION TECHNIQUES: BOOTSTRAPPING AND PARTIAL EVALUATION

Maria Paola Bonacina

Dipartimento di Scienze dell'Informazione

Università degli Studi di Milano

RIASSUNTO

Questo lavoro presenta due tecniche di generazione di programmi: generazione di compilatori mediante bootstrapping e specializzazione di programmi mediante valutazione parziale. I due metodi sono descritti e poi confrontati nell'applicazione alla generazione di compilatori. La notazione dei T-diagrams, tradizionalmente applicata a descrivere processi di bootstrapping, viene estesa in questo articolo alla valutazione parziale.

ABSTRACT

This work presents two program generation techniques: compiler generation by bootstrapping and program specialization by partial evaluation. The two methods are described and then compared on the common ground of compiler generation. The T-diagram notations traditionally applied to describe bootstrapping processes is extended to partial evaluation as well.

1. INTRODUCTION

Automated program generation should allow in principle to reduce the amount of hand written code. Two approaches to program generation are program synthesis and program transformation.

Program synthesis consists in automatically deriving a program from a hand written specification. It generally requires a specification language and a compiler being able to translate a specification into a program in some executable language.

Program transformation consists in deriving from a given program a new one. It generally requires a program transforming program being able to obey suitable transformation directives. A program transformation technique is usually requested to be semantic preserving: the new program semantic should be some way related to that of the original program.

This paper is devoted to two classes of program transformation techniques: bootstrapping based techniques and partial evaluation based techniques. These techniques are both very powerful because they do not require explicitly given directives.

Bootstrapping techniques are special purpose techniques: they allow to derive high cost compilers from low cost compilers and interpreters [1,8]. A high cost compiler is for instance a compiler from a high level language to machine language, written in machine language. A low cost compiler is for instance a compiler from a high level language to an intermediate language, written in the high level language itself.

Partial evaluation techniques are general purpose techniques: they allow to derive specialized programs from given programs by evaluation with respect to partial data [3,9]. Although general purpose in principle, partial evaluation is especially well suited to work on meta-programs, i.e. programs taking as input other programs. This is the case of interpreters and compilers, so that partial evaluation looks a promising technique in the same domain where bootstrapping is applied [5,4].

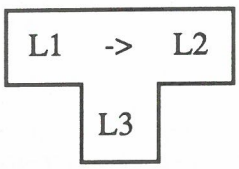
The paper is structured as follows: first bootstrapping techniques and partial evaluation are reviewed. Bootstrapping processes and partial evaluation processes are represented by T-diagrams and L-diagrams respectively.

Next the connections between the two techniques are explored. Since bootstrapping is a special purpose technique, the comparison is carried out focusing on the application of partial evaluation to compiler generation. The key concept is self-application: bootstrapping exploits self-compilation and partial evaluation exploits self-projections to generate compilers.

2. REPRESENTING TRANSLATORS

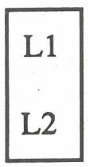
In order to describe compiler generation techniques we introduce the classical graphic *T-diagrams* notation [2]. The T-diagram name derives from the form of the basic box:

Translator or Compiler:



L_1 is the source language, L_2 is the object language and L_3 is the language the compiler is written in.

Interpreter:



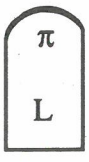
representing the interpreter of L_2 written in L_1.

Virtual Machine:

representing the virtual machine whose language is *M*.

In the original T-diagrams notations the above triangular notation represents just real machines, not virtual ones. We extend it to represent virtual machines in order to give the language a greater abstraction power: a single triangular diagram or represents a virtual machine obtained composing as many software levels as needed on top of some hardware.

Program:



representing the program π written in language *L*.

3. BOOTSTRAP TECHNIQUES

The bootstrap technique is a technique to generate compilers.

The key concept is self-application or, more precisely, self-compiling [8]: a compiler written in a high level language is translated into a lower language and then is given to its translation.

The two basic bootstrap schemes are *half bootstrap* and *full bootstrap* Let's fix some naming conventions:

L: source language (generally a high level language)

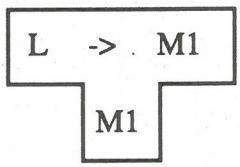
O: intermediate language (generally a language whose level is lower than that of *L*)

M_1: machine language for machine 1

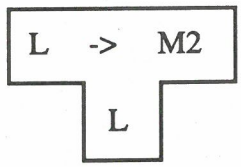
M_2: machine language for machine 2.

Basic half bootstrap scheme:

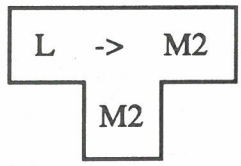
Given the compiler from *L* to M_1 written in M_1,



and having written in *L* the compiler from *L* to M_2,

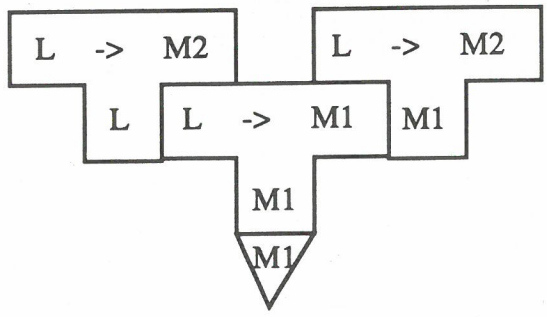


the compiler from *L* to M_2 written in M_2,



is obtained by the following steps:

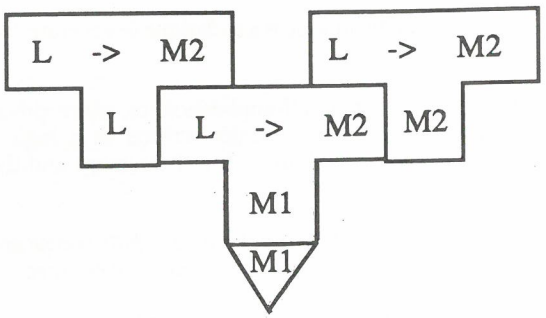
1.



this is the cross compiler running on machine M_1 and ge-

nerating code for machine M_2.

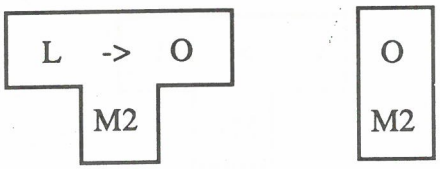
2.



which is the final result. This technique is called half-boostrap because it is executed on the source machine M_1 requiring cross compilation. The next one is called full boot-strap because the whole process is executed on the target machine M_2 [1].

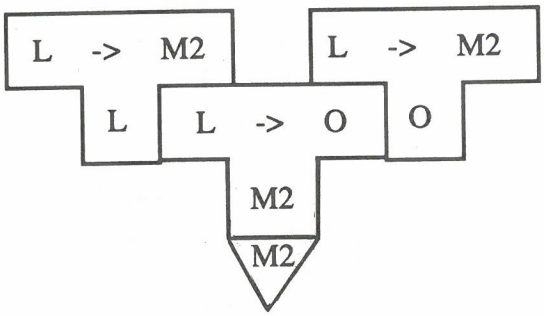
Basic full bootstrap scheme:

Given the compiler from L to O written in M_2 and the interpreter of O written in M_2,

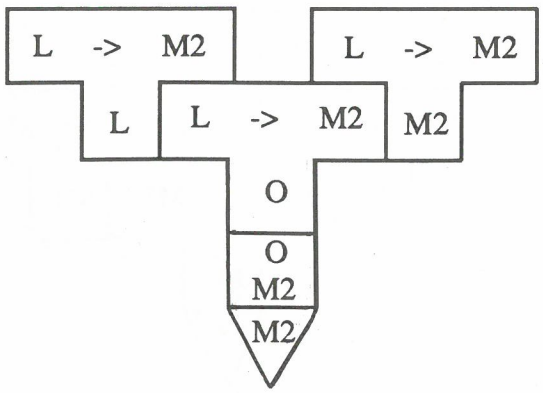


and having written in L the compiler from L to M_2, the compiler from L to M_2 written in M_2 is derived by the following steps:

1.



2.



This application requires that the initially given compiler from L to O written in M_2, though unsatisfactory, is able to compile the compiler written in L [1].

4. PARTIAL EVALUATION

This paragraph presents partial evaluation as defined in [3]. Let D be a data domain, L and O two programming languages: as a special case L and O can be the same language. In the following lower case greek letters are programs written in L and lower case latin letters are programs written in O: π: $D^n \rightarrow D^n$ is a program in L, p: $D^n \rightarrow D^n$ is a program in O.

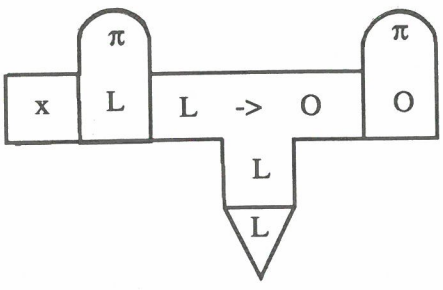
Definition: the *partial evaluation of a program* $\pi \in L$ with respect to a partial input tuple x is a *residual program* $p_x \in O$ such that $\forall y$, $\pi(x,y) = w$ if and only if $p_x(y) = w$: in short peval$(\pi,x) = p_x$, where *peval:* $L \times D^k \rightarrow O$ [3].

Partial evaluation is an intermediate process between interpretation and compilation. A partial evaluator works as follows: it interprets the given program, executing it on the available data. However since this data set is not complete, there are statements that cannot be executed: for instance conditional statements where the test cannot be answered for lack of data. These statements are suspended. All statements depending from a suspended one get suspended as well. Suspended instructions are translated and output as residual program. This residual program is a specialization of the original one in the sense given by the definition of partial evaluation itself: the residual program feeded with some input will yield the same result as the original program would, if applied to this input and that one which drove the specialization. The data given at partial evaluation time gets embedded in the residual program, because the residual program is what is left to be done working precisely on that data.

5. REPRESENTING PARTIAL EVALUATION BY T-DIAGRAMS

When L and O are two different languages partial evaluation embeds a translation process which can be represented by a T-diagram. However the T-diagram notation needs to be extended in order to represent also data given at partial evaluation time. Data can be represented by a box on their own:

$$peval(\pi,x) = \pi_x$$



The T-diagram representing the partial evaluator differs from that representing the compiler in that the first one gets concatenated on its left side to a diagram given by the composition of two diagrams, a program diagram and a data diagram. When the partially evaluated program is a meta-program, the data box gets instantiated to a program, so that the notation is completely homogeneous, being every element a program. It is the case in the following application of this notation to represent compiler generation by partial evaluation.

6. BOOTSTRAPPING AND PARTIAL EVALUATION

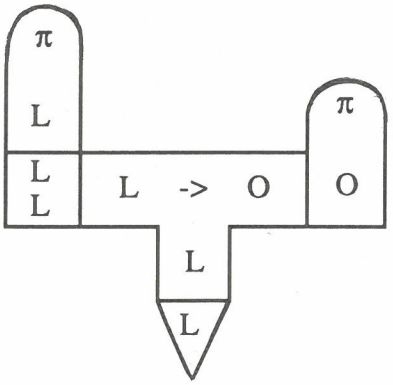
A self-applicable partial evaluator is called autoprojector. Successive projections of a autoprojector onto itself generate object programs, compilers and compiler-compilers as shown first by Futamura [5]. The first condition to get an autoprojector is writing *peval* in the same language L it interprets.

Let's suppose an interpreter *int*, a partial evaluator *peval* for L, both written in L, and a virtual machine for L are available.

First projection:

$peval(int, \pi) = int_\pi$ such that $int(\pi,x) = y$ if and only if $int_\pi(x) = y$ by definition of *peval*.

Since *int* is the interpreter, $int(\pi,x) = y$ if and only if $\pi(x) = y$. It follows that $\pi(x) = y$ if and only if $int_\pi(x) = y$. That means int_π is the translation of π in the language O, obtained partially evaluating or projecting the interpreter *int* onto π:

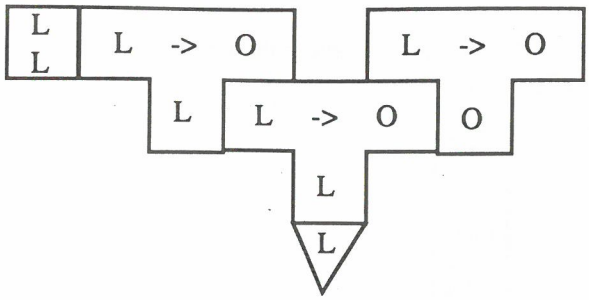


Our extended T-diagrams formalism is exploited giving as input to the partial evaluator a pair of programs, where one is the input to the other. In this case the pair is given by the interpreter *int* and the program π. The partial evaluation process looks in the diagrams as collapsing two occurrences of the language L into one and translating it into O.

Second projection:

$peval(peval,int) = peval_{int}$ such that for all programs π, written in L, $peval(int,\pi) = peval_{int}(\pi)$ by definition of *peval*.

Since $peval(int,\pi) = int_\pi$, i.e. the translation of π in the language O, by the first projection, $peval_{int}$ is the compiler from L to O:

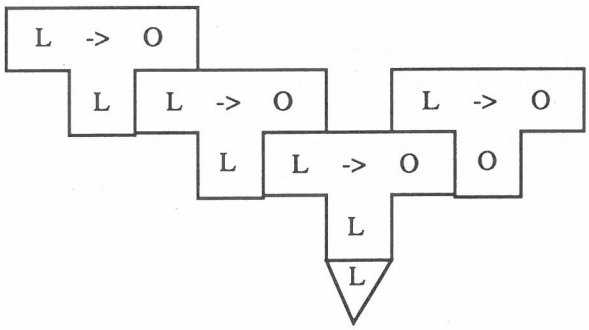


Again a collapse and a translation occur in the diagram.

Third projection:

$peval(peval,peval) = peval_{peval}$ such that for all interpreter *int* written in L $peval(peval,int) = peval_{peval}(int)$ by definition of *peval*.

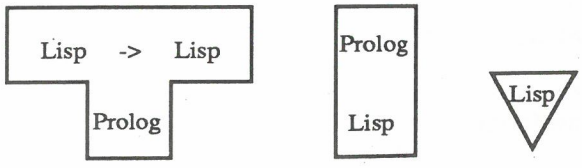
Since $peval(peval,int) = peval_{int}$ is the compiler from L to O by the second projection, $peval_{peval}$ is the compiler-compiler, transforming an interpreter into a compiler:



It has to be noticed that using T-diagrams does not allow to distinguish between a compiler and a compiler-compiler, because only involved languages are emphasized.

Example:

This example shows the application of the extended T-diagrams notation to the application of partial evaluation described in the title of [7]. Let *peval* be a partial evaluator for Lisp written in Prolog, p-int a Prolog interpreter written in Lisp and a Lisp virtual machine be given. These elements are represented in this order as follows:

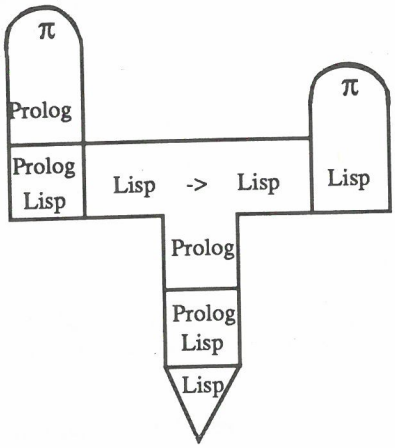


The goal is to get a Prolog compiler: it is impossible to

compute *peval(peval,p—int),* because *peval* expects a Lisp program, while it is written in Prolog. It is not an autoprojectot: an autoprojector written in Lisp is required.

First projection (general case):

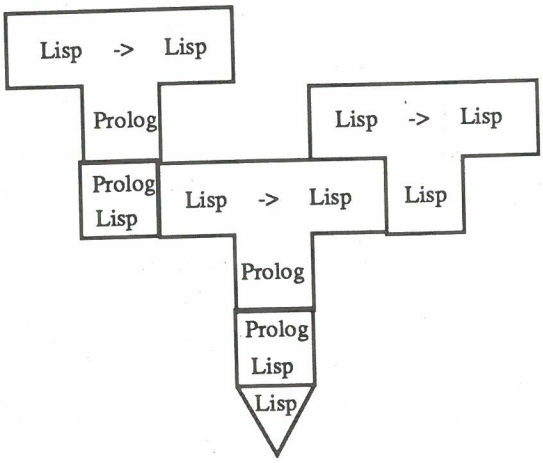
peval(p—int,π*)* = *p—int*$_\pi$ is the Lisp translation of the Prolog program π:



First projection (special case):

peval is itself just a Prolog program so it can be substituted to π in the above projection.

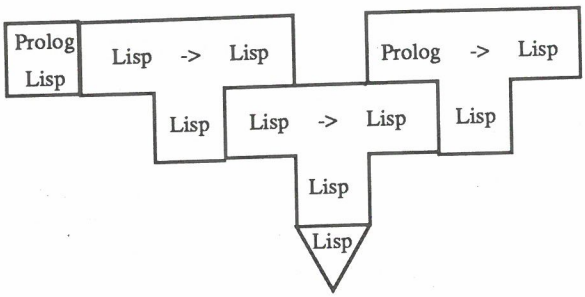
peval(p—int,peval) = *p—int*$_{\text{peval}}$ is the partial evaluator for Lisp written in Lisp, i.e. the autoprojector; let's call it *l—peval:*



Second projection:

Now the autoprojector is available and it can be given to itself with the interpreter to yield the compiler:

l—peval(l—peval,p—int) = *l—peval*$_{\text{p-int}}$ is the Prolog to Lisp compiler written in Lisp:



7. ACKNOWLEDGEMENTS

I sincerely thank Giovanni Degli Antoni for useful suggestions.

8. REFERENCES

[1] G. Castelli, G. Rossi, LA PORTABILITÀ DEL LOGICALE SU PICCOLI SISTEMI, Technical Report, Istituto di Cibernetica, Università degli Studi di Milano, Italy

[2] J. Earley, H. Sturgis, A FORMALISM FOR TRANSLATOR INTERACTIONS, CACM, Vol. 13, No. 10, 607-617, October 1970

[3] A.P. Ershov, ON THE PARTIAL COMPUTATION PRINCIPLE, Information Processing Letters, Vol. 6, No. 2, 38-41, April 1977

[4] H. Fujita, K. Furukawa, A SELF-APPLICABLE PARTIAL EVALAUTOR AND ITS USE IN INCREMENTAL COMPILATION, In D. Bjorner, A.P. Ershov, N.D. Jones (eds.), Proc. Workshop on Partial Evaluation and Mixed Computation, Gl. Avernæs, Denmark, North Holland, to appear, October 1987

[5] Y. Futamura, PARTIAL EVALUATION OF COMPUTATION PROCESSES, An Approach to a Compiler-Compiler Systems, Computers, Controls, Vol. 2, No. 5, 45-50, 1971

[6] Ghezzi, Jazayeri, PROGRAMMING LANGUAGE CONCEPTS, J. Wiley & Sons, New York, 1982

[7] K. Khan, A PARTIAL EVALUATOR OF LISP WRITTEN IN A PROLOG WRITTEN IN LISP INTENDED TO BE APPLIED TO THE PROLOG AND ITSELF WHICH IN TURN IS INTENDED TO BE GIVEN TO ITSELF TOGETHER WITH THE PROLOG TO PRODUCE A PROLOG COMPILER, UPMAIL, Dept. of Computing Science, Uppsala University, Sweden, March 1982

[8] O. Lecarme, M.C. Peyrolle-Thomas, SELF-COMPILING COMPILERS: AN APPRAISAL OF THEIR

IMPLEMENTATION AND PORTABILITY, Software Practice and Experience, Vol. 8, No. 2, 149-170, 1978

[9] L.A. Lombardi, INCREMENTAL COMPUTATION, In F.L. Alt and M. Rubinoff, eds., Advanced in Computers, Vol. 8, 247-333, Academic Press, 1967

[10] P. Sestoft, A.V. Zamulin, LITERATURE LIST FOR PARTIAL EVALUATION AND MIXED COMPUTATION, In D. Bjorner, A.P. Ershov and N.D. Jones eds., Proc. Workshop on Partial Evaluation and Mixed Computation, Gl. Avernæs Denmark, North Holland, to appear, October 1987

ARTIFICIAL INTELLIGENCE AND CREATIVITY: A PROPOSAL FOR AN AUTOMATION INVENTING NEW OBJECTS

Rita Pizzi, Donatella Galli

Dipartimento di Scienze dell'Informazione

Università di Milano

RIASSUNTO

L'articolo descrive un "automa creativo", ossia una procedura che produce la descrizione di oggetti originali ed innovativi a partire da un dominio di oggetti reali già esistenti. La procedura adottata per generare nuovi oggetti viene presentata attraverso una dettagliata formalizzazione del dominio concettuale nel quale l'automa opera dell'insieme di regole euristiche utilizzate. Infine un esempio completo mostra a quali risultati l'automa è in grado di prevenire, sottolineandone gli spunti più innovativi e creativi.

ABSTRACT

This paper describes a "creative automaton" that is a procedure which produces new, original objects from a domain of real, already existing ones. An intuitive presentation of a procedure used to generate objects is given, followed by a detailed formalization of the conceptual domain where the automation operates and of the set of heuristics rules it uses to perform its tasks. A final example shows some results which can be obtained by the automaton emphasizing their creative and innovatory contents.

1. INTRODUCTION

The famous remark by Thomas Edison, "Genius is 1 percent inspiration and 99 percent perspiration", reflects the commonplace that, in spite of the vagueness and mistery it has been surrounded, human creativity can be, at least partly, explained and reproduced.

Nevertheless, creativity is still a very complex matter of study because it hasn't been exactly defined, yet.

Creativity has been analyzed from different points of view by psycologists, cognitive scientists and A.I. people who sometimes show very different opinions about it.

N. Chomsky's [5] definition of creativity for language as a finite set of rules generating an infinite variety of instances has surely a lot of supporters. But even though it can well explain the productive aspects of creativity [18], it cannot clarity the very feature which distinguish it from general problem solving, that is the ability to vary the very structure of rules, when they are found inadequate.

Many different authors, such as M. Boden [1], A. Bundy [2, 3, 4], and M. Yazdany [18], underline the importance of standard rules violation and meta__rules generation. Yazdany [18] proposes a multi__level universe where level 0 contains object, level 1 contains rules for generating objects of domain 0, level 2 contains rules for generating rules of domain 1 and so on.

G. Polya [14, 15] points out that finding a solution to a problem should mark not the end, but only the beginning of an enquiry: it should be followed by attempts to find rule-set used to solve it, in order to explore possible extensions or refinements to it.

According to H.A. Simon [12, 13], creativity is just a particular kind of problem solving where the problem is particularly hard to formulate and the results are new and unconventional. It follows that even in this case creativity is pursued with changes to the solution generator consisting in abstraction from details and unintuitive rules.

D.R. Hofstadter [7, 8] mantains that "the crux of creativity lies in the ability to manufacture variations on a the-

me" of existing concepts as each concept had "some kind of knobs on it" and twiddling them one could modify the object parameters. He agrees, too, that the most evolved mental activities, among which creativity, result from repeated applications of cognitive levels and meta__levels.

However the most concrete contribution to research about creativity may have been given by D.B. Lenat [9, 10, 11]. He supports the heuristitc structure of creative processes. Besides the peculiar meaning in a particular field, Lenat's heuristic can be summed up in a "generalization" principle (search and abstraction from data), and a "specialization" principle (search for exception to the very rules). Moreover the "analogy" principle deals with the excerption of new rules out of the body of the previously established ones. This principle, once more showing the existence of meta__rules governing the creative processes, constitutes a basis for the "Eurisko" program, an extension to A.M. - a program discovering new mathematical concepts - whic is able to create new heuristic rules modifying its own heuristics.

Far from pretending to be a contribute to cognitive sciences, this paper aims both to investigate the possibility of supporting human creativity and to cast a side glance at "creativity in large" through the formalization of a very restricted field ("creativity in small"), easily communicable and liable of further extensions. About this matter, Lenat [11] has already shown evidences that the efficiency of computational models of creative processes is inversely proportional to the problem space extension.

This work has also been conformed to the actual growing interest in the integration of different functions into single objects. To this respect, a "creative automaton" has been implemented (using a Prolog interpreter) which produces new objects, models of through original compositions of already existing ones. The automation structures an universal set of objects and operates on it by means of a set of heuristic rules which have some analogies with Lenat's heuristics; such as looking up to the extreme connections between variables and using meta__rules and cognitive levels overlapping in order to obtain the best results.

In various environments the current opinion (see Prigogine [16, 17]) is that impredictable and innovatory properties emerge, in nature, at the borders of the examined domains and that the growing complexity of those domains can arrange for the birth of properties of this kind.

The PROLOG implementation, wholly isomorphic to the heuristic rules formalized in what follow, is structured as a cognitive model.

A "focus of attention" scans a configuration of semantic primitives (that is concepts, seen as the power set of the configuration) according to a selective criterion: only a framing showing loci of maximum continuity or maximum discontinuity in the concept space can get through the attentional threshold.

Therefore the "focus of attention" looks for **innovatory**

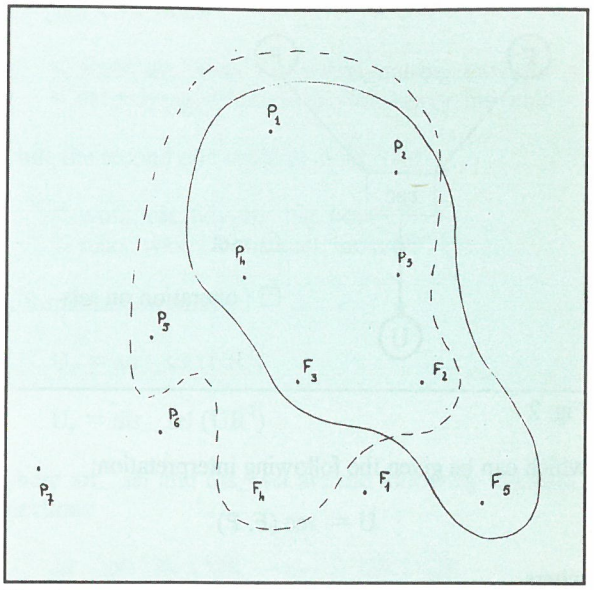


Fig. 1

readings of the configuration of semantic primitives shifting through concepts which are bounded by maximum likeness (association) or maximum distinction.

The following heuristics constitute suitable arrangements of these movements.

The picture above is an example of a configuration of semantic primitives and of the likeness between concepts, where the P's indicate the properties and the F's the functions of the concept, as explained later.

The following paragraph contains the formalization of the model, interleaved to a, step by step, intuitive presentation which allows an easier interpretation of the formalism. Each step is also fitted out with an illustrating example. The last paragraph contains a whole example and a short discussion.

2. THE MODEL

2.1 The universal set of objects U

This basic idea is that a real object can be individuated by its physical properties and the functions it can perform.

Taking this approach, a set of predefined functions F and a set of predefined properties P are given in order to build a universe U of real objects. In agreement with the "generalization principle" of Heuristics [11], the "Universal Set of Objects" U is recursively generated starting from minimal objects which are individuated by just one function and one property.

The process can be sketched in the data-flow schema below:

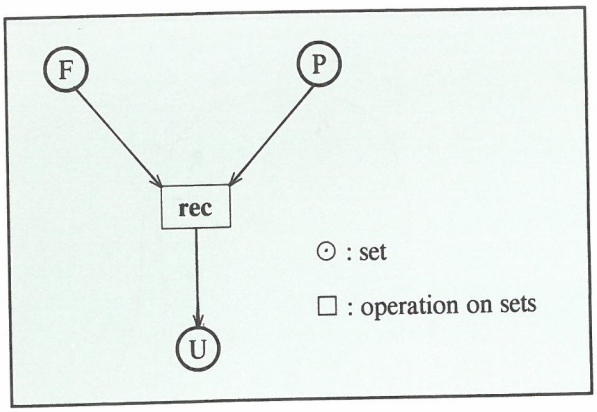


Fig. 2

which can be given the following interpretation:

$$U = \text{rec}\ (F, P)$$

where

$$\text{rec: } 2^F \times 2^P \longrightarrow 2^F \times 2^P$$

P can be partitioned into two isomorphic subsets P1 and ~P1 where ~P1 elements are properties which contradict properties in P1.

According to the following conventions:

$<a_1, \ldots, a_n>$ denotes set whose elements are $a_1, \ldots, a_n$

$<\alpha \mid \beta>$ denotes a set where α and β are subsets of two different sets

$\alpha + \beta$ denotes the usual set union of sets α and β

$\alpha * \beta$ denotes the usual set intersection of sets α and β

$\alpha - \beta$ denotes the usual set complement of set β to set α

$\alpha \odot \beta$ denotes $(\alpha + \beta) - (\alpha * \beta)$ where operation * has a higher precedence on operators + and —

$FO = F + <f_0>$ where f_0 is the "null function"

$PO = P + <p_0>$ where p_0 is the "null property"

$$\text{rec}\ (<f_i>,<p_j>) = <f_i \mid p_j> \qquad f_i \in F \qquad p_i \in P$$

$$\text{rec}\ (<f_1, \ldots, f_{k+1}>,<p_1, \ldots, p_{l+1}>) = \text{rec}\ (<f_1, \ldots, f_k>,<p_1, \ldots, p_l>) + <f_{k+1} \mid p_{l+1}>$$

where $f_1, \ldots, f_{k+1} \in F_0$; $p_1, \ldots, p_{l+1} \in P_0$; $f_{k+1} \notin <f_1, \ldots, f_k>$; $p_{l+1}, \sim p_{l+1} \notin <p_1, \ldots, p_l>$

For example, given:

F = < work, relax, tidy up, wash, eat >

P = < heavy, movable, electronical, big, horizontal, not-heavy, not-movable, not-electronical, not-big, not-horizontal >

tipical objects in U are the following:

< wash, work | big, heavy, movable >

< tidy up, wash | electronical, not-heavy, movable >

. . . . . .

Intuitively the construction of the universe U corresponds to the application of the heuristic:

ABSTRACT GENERALIZATION RULE: build recusively a "universal set of objects" starting from minimal objects defined by just one property and one function.

2.2 The "Reduced Universe Set" UR

Successively, the automaton gives the user the opportunity to specify the minimal characteristics (properties and functions) of the object to be produced. It then removes from universe U all objects which are unconsistent with user specification.

This operation defines a "REDUCED UNIVERSE SET" UR according to the following heuristic rule:

FOCUSING RULE: reduce the universe erasing objects not containing at least a required function or containing properties which are complementary to the requested ones.

In the notation already used:

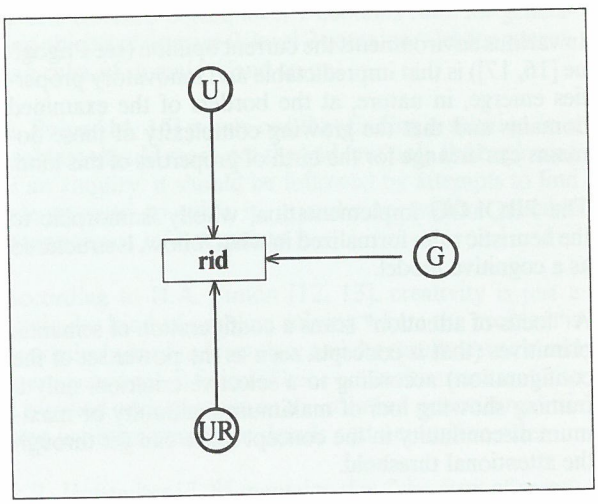


Fig. 3

where, if τ is the user request:

$$\tau \in G,\ G \subseteq U.$$

and the "REDUCED UNIVERSAL SET" UR can be defined as follows:

$$UR = \text{rid}\ (U,I)$$

where, according to Zermelo's choice axiom a "selection function" exists

$$\text{rid}\quad U \times G \longrightarrow U$$

$$\text{rid}\quad (\alpha,\tau) = \alpha \qquad \text{if } \alpha \in U,$$

$$\alpha = < \alpha_f \mid \alpha_p >,$$

$$\tau = < \tau_f \mid \tau_p >,$$

$$\alpha_f * \tau_f = <>,$$

$$p_i \in \alpha_p \qquad \sim p_i \notin \alpha_p$$

According to this rule, if the following minimal characteristics have been specified:

< tidy up, relax | not heavy >

has been specified, among the removed objects will be:

< eat, wash | movable >
< tidy up, relax | heavy > and so on.

while the set UR will contain objects of the form:

< tidy up, eat | movable, electronical >
< relax, eat, wash | horizontal, not-big >

This definition of the set UR has meaning of delimitating the problem space to objects which are someway related to the request by the user.

2.3 The new object's generation

Once the set UR has been determined, the automaton operates associations and dissociations among its elements following a set of heuristics rules outlined below:

ANALOGY-SELECTION RULE: select all pairs of objects sharing properties and functions so that common elements predominate over different ones.

DISTINCTION-SELECTION RULE: select all pairs of objects which don't share any common characteristics.

These couple of rules produce two sets Ux and Uy containing respectively pairs of objects which have a maximum analogy or haven't any. A result of the application of the first rule is the pair:

< work, eat, wash | horizontal, not-big, movable >
< eat, tidy-up, wash | horizontal, heavy, movable >

while the second one could give the pair:

< work, eat, tidy-up | big, heavy >
< relax, wash | horizontal, movable >

A formalization can be:

$$U_x = \text{an__sel}\ (UR^2)$$

$$U_y = \text{dis__sel}\ (UR^2)$$

where an__sel and dis__sel are the following selection functions:

an__sel: UR x UR ——> UR x UR

dis__sel: UR x UR ——> UR x UR

with

$$\text{an__sel}\ (\alpha,\beta) = (\alpha,\beta) \text{ if } 2 \mid \alpha * \beta \mid \geq \mid \alpha \cap \beta \mid$$

and

$$\text{dis__sel}\ (\alpha,\beta) = (\alpha,\beta) \text{ if } \alpha * \beta = <>$$

Schematizing:

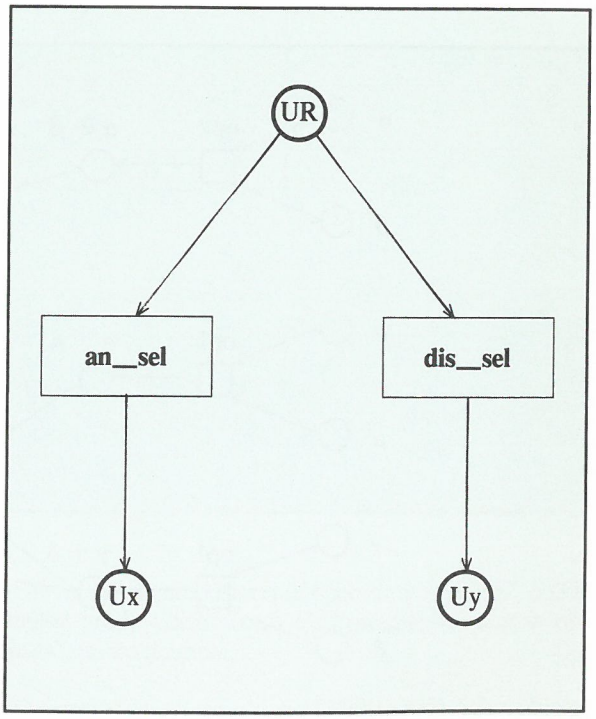


Fig. 4

Now, pairs of objects in sets Ux and Uy are combined according to the following:

ANALOGY-ASSOCIATION RULE: combine pairs of objects selected by analogy-selection rule.

ANALOGY-DISTINCTION RULE: synthesize new objects from different elements in pairs of analogous objects.

DISTINCTION-ASSOCIATION RULE: combine pair of wholly distinct objects.

The last two rules, in particular, search for the maximum creativity associating objects in an unusual way.

A similar heuristic has been used by D.B. Lenat in its A.M. [9].

The above rules are accomplished by the following operators:

op1: U_x ——> U where $x = <\alpha,\beta>$ and $op_1(x) = \alpha+\beta$
op2: U_x ——> U where $x = <\alpha,\beta>$ and $op_2(x) = \alpha \odot \beta$
op3: U_x ——> U where $y = <\alpha,\beta>$ and $op_3(y) = \alpha+\beta$

where op1 accomplishes the analogy-association rule,
op2 accomplishes the analogy-distinction rule,
op3 accomplishes the distinction-association rule.

Applied to the pairs of objects which where selected before the operators just defined produce:

op1 (<work, eat, wash| horizontal, not-big, movable>, <eat, tidy-up, wash| horizontal, heavy, movable>) =
= <work, eat, wash, tidy-up| not-big, horizontal, movable, heavy>

op2 (<work, eat, wash| horizontal, not-big, movable>, <eat, tidy-up, wash| horizontal, heavy, movable>) =
= <work, tidy-up| not-big, heavy>

op3 (<work, eat, tidy-up| big, heavy>, <relax, wash| horizontal, movable>) =
= <work, eat, tidy-up, relax, wash| big, heavy, horizontal, movable>

The object obtained by the application of the association and dissociation rules seen above are joined to the objects in UR and the obtained set is submitted to the application of the same rules. This process is repeated a prefixed number n of times in order to gain the highest complexity in the results.

That behaviour of the automaton agrees with a principle of "complexity" [7] stating that the greatest complexity can give raise to the most meaningful results.

The heuristic supporting this process is the following:

INVERTED OCKHAM'S RAZOR RULE: keep on combining objects which derive from the application of lower level rules, using the same rules in order to obtain the maximum level of complexity.

A tipical sequence of computation can be represented in the diagram below.

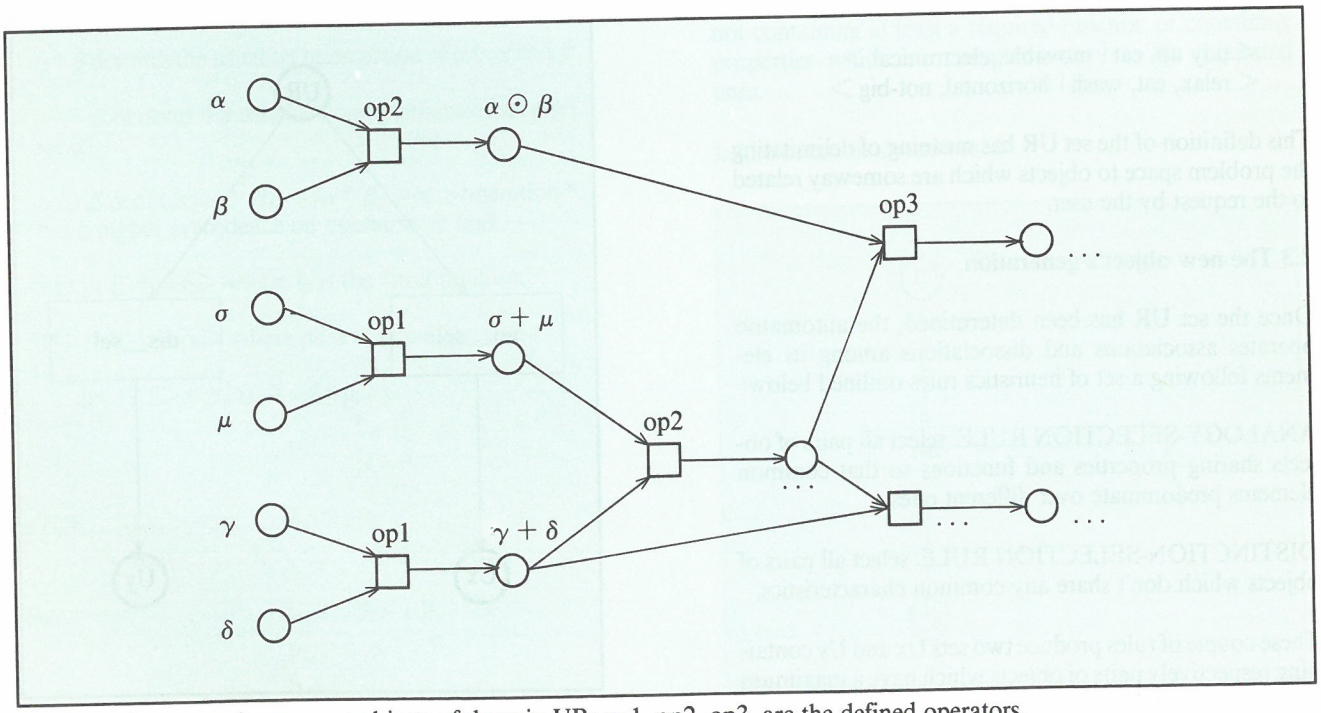


Fig. 5 α, β, σ, μ, τ, δ represent objects of domain UR; op1, op2, op3 are the defined operators.

The notation:

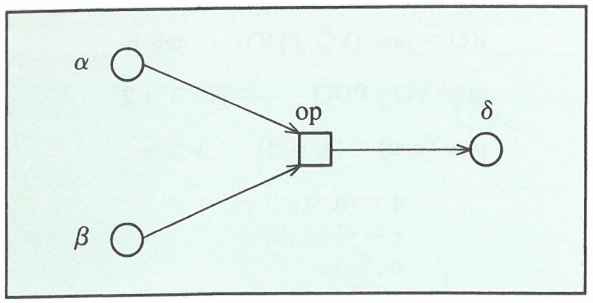


Fig. 6

means: $op(\alpha,\beta) = \delta$

As it is shown in the graph above, objects can be combined, no matter if they belong to different level of complexity. The global result will be chosen as a set SO_{MAX} containing the most complex objects obtained.

The following diagram describes the whole:

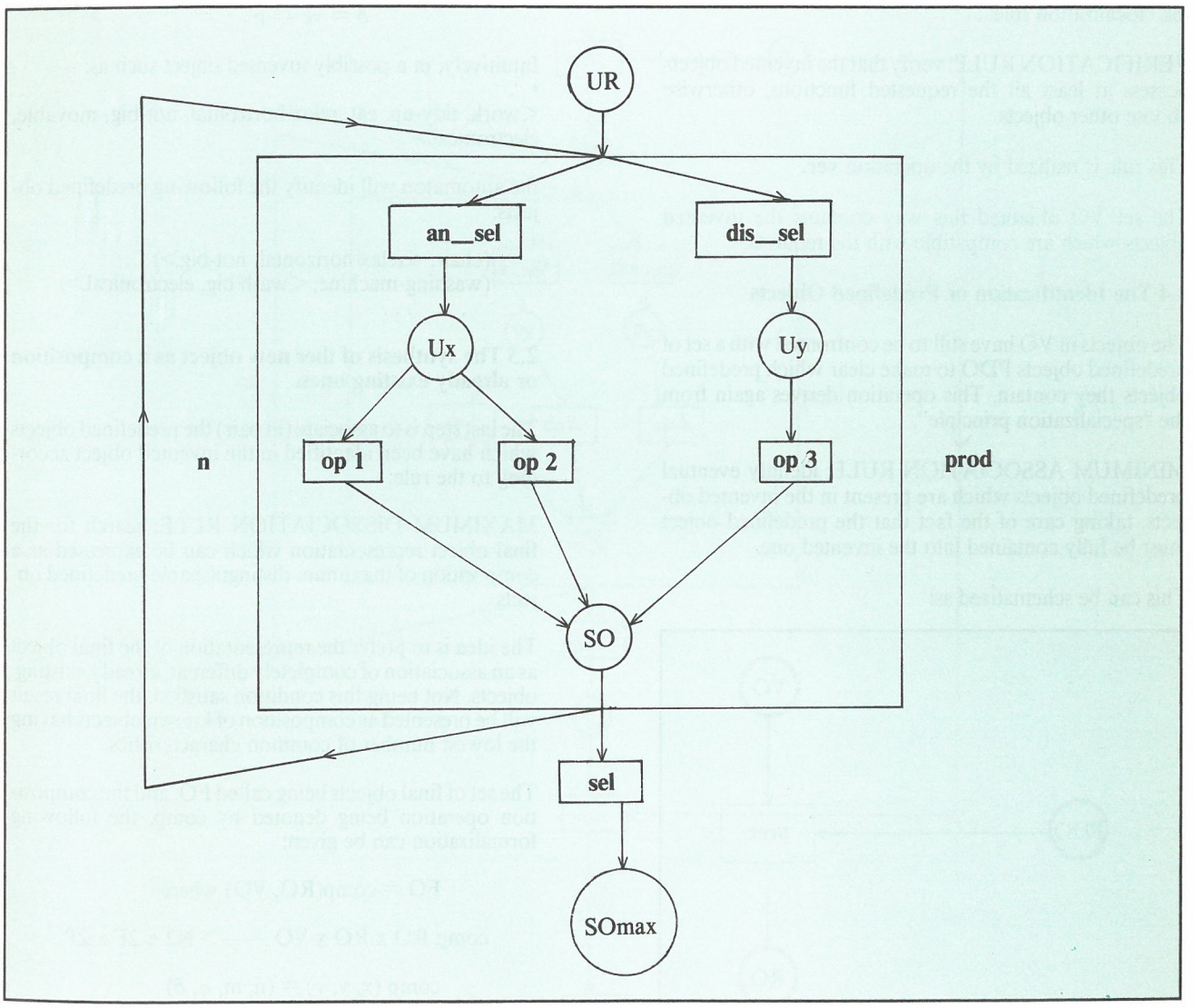


Fig. 7

The meaning of UR, an__sel, dis__sel, Ux, op1, op2, op3, should already be clear.

SO denotes the set of objects which are geherated by whichever composition of operations op1, op2, op3 applied to objects in UR.

The repeated composition of operations op1, op2, op3 is called prod, which stands for "production of new objects", in the diagram.

The operation sel (selection of the most complex object) extracts froms the set SO the objects obtained by the most

complex application of rules to objects in UR.

In the previous diagram SO is the set of all objects denoted with a ⊙, while SO_{MAX} contains just the two right most objects.

The level k of the objects selected by operation sel may be lower than n. This happens when an__sel and dis__sel find sets U_x or U_y which are empty.

Once SO_{MAX} has been calculated, its elements must bo tested in order to verify if they really satisfy the minimal requests specified by the user. The principles followed here is analogous to the "specialization principle" used in the "focalization rule".

VERIFICATION RULE: verify that the invented objects possess at least all the requested functions, otherwise choose other objects.

This rule is realized by the operation **ver.**

The set VO obtained this way contains the invented objects which are compatible with the requests.

2.4 The Identification or Predefined Objects

The objects in VO have still to be confronted with a set of predefined objects PDO to make clear which predefined objects they contain. This operation derives again from the "specialization principle".

MINIMUM ASSOCIATION RULE: identify eventual predefined objects which are present in the invented objects, taking care of the fact that the predefined object must be fully contained into the invented one.

This can be schematized as:

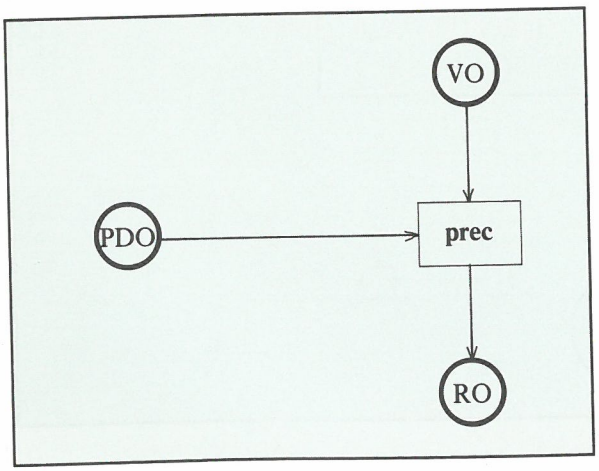


Fig. 8

Set PDO has been defined as a set of elements of the form:

(n, α) where $n \in N$, N is a set of names identifying objects

$\alpha \in SU$, SU is a subset of U.

The formal interpretation of the schema given is the following:

$$RO = prec\ (VO, PDO) \qquad \text{where}$$

$$prec: VO \times PDO \longrightarrow N \times 2^F \times 2^P$$

$$prec\ (\tau, u) = (n, \sigma, d) \qquad \text{where}$$

$$u = (n, <\pi_a \mid \pi_b>)$$
$$\tau = <\tau_a \mid \tau_a>$$
$$\pi_b \subseteq \tau_a$$
$$\pi_b \subseteq \tau_b$$
$$\sigma = \tau_a - \pi_a$$
$$\delta = \tau_b - \pi_b$$

Intuitively, in a possibly invented object such as:

<work, tidy-up, eat, relax|horizontal, not-big, movable, electronical>

the automaton will identify the following predefined objects:

(chair, <relax|horizontal, not-big>)
(washing-machine, <wash|big, electronical>)
... ...

2.5 The synthesis of ther new object as a composition or already existing ones.

The last step is to associate (in pair) the predefined objects which have been identified in the invented object according to the rule:

MAXIMUM DISSOCIATION RULE: search for the final object representation which can be expressed as a composition of maximum distinguishable predefined objects.

The idea is to prefer the representation of the final object as an association of completely different, already existing, objects. Not being this condition satisfied, the final result will be presented as composition of known objects having the lowest number of common characteristics.

The set of final objects being called FO, and the composition operation being denoted by comp, the following formalization can be given:

$$FO = comp(RO, VO) \text{ where}$$

$$comp\ RO \times RO \times VO \longrightarrow N2 \times 2F \times 2P$$

$$comp\ (x, y, \tau) = (n, m, \sigma, \delta)$$

with; $x, y \in RO$
$\tau = <\tau_a \mid \tau_b>$
$\sigma = \tau_a - (\alpha_a + \beta_a)$
$\delta = \tau_b - (\alpha_b + \beta_b)$

In the example above the final result will be presented as composition of "chair" and "washing-machine", which are a pair of objects having nothing in common.

2.6 The Procedure as a Whole

Finally, the whole procedure can be represented in the following schema:

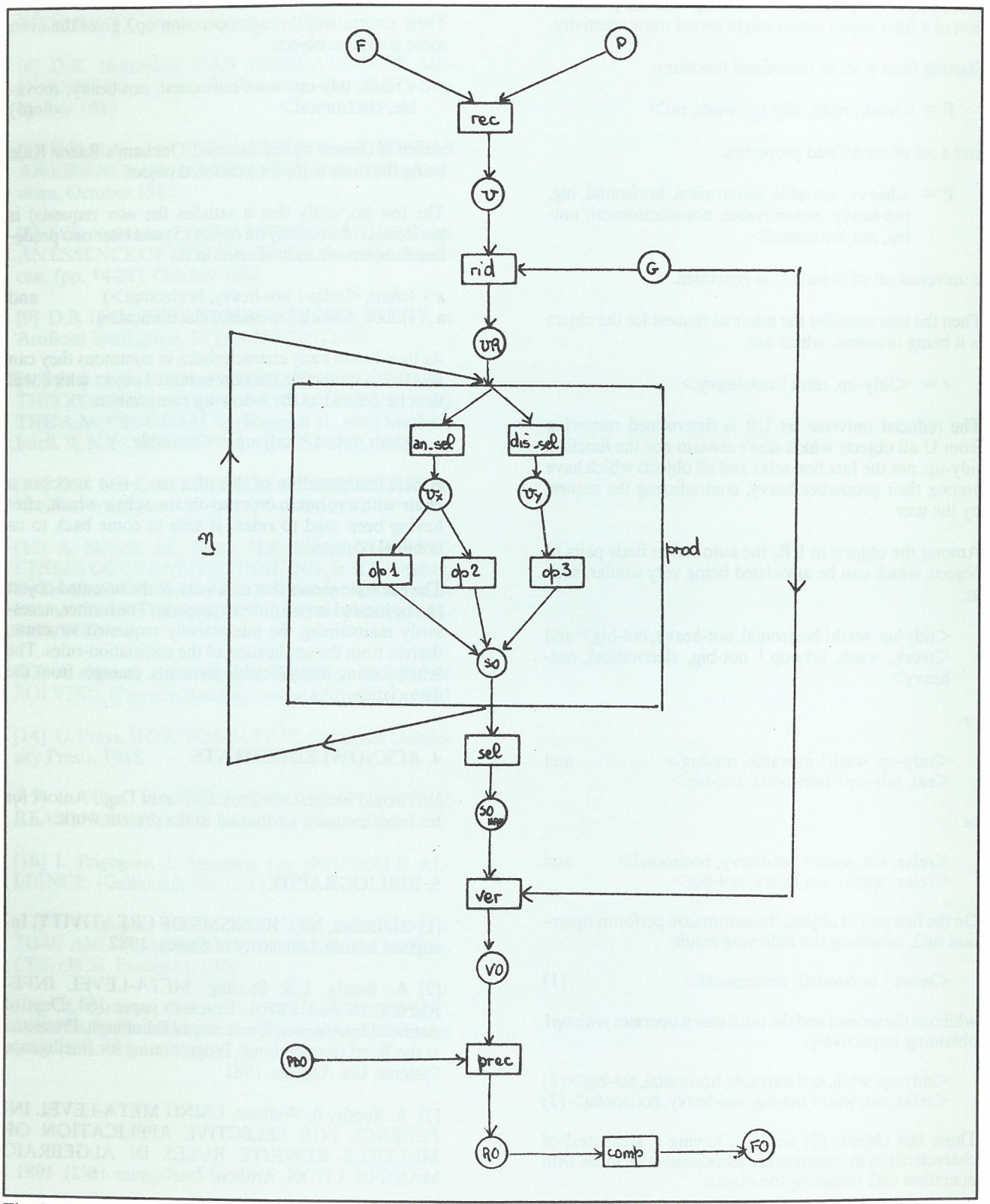


Fig. 9

3. AN EXAMPLE

In the following, a whole example will be presented of the execution of the procedure. It ends up with the presentation of a final object which might reveal some creativity.

Starting from a set of predefined functions:

F = <work, relax, tidy-up, wash, eat>

and a set of predefined properties:

P = <heavy, movable, electronical, horizontal, big, not-heavy, not-movable, not-electronical, not-big, not-horizontal>

a universal set of objects U is generated.

Then the user specifies the minimal request for the object is it being invented, which are:

τ = <tidy-up, relax | not-heavy>

The reduced universe set UR is determined removing from U all objects which don't contain nor the function tidy-up, nor the function relax and all objects which have among their properties heavy, contradicting the request by the user.

Among the objects in UR, the automaton finds pairs of objects which can be associated being very similar, such as:

<tidy-up, wash | horizontal, not-heavy, not-big> and <work, wash, tidy-up | not-big, electronical, not-heavy>

or

<tidy-up, wash | movable, not-big> and <eat, tidy-up | horizontal, not-big>

or

<relax, eat, wash | not-heavy, horizontal> and <relax, wash | not-heavy, not-big>

On the first pair of objects the automaton performs operation op2, obtaining the following result:

<work | horizontal, electronical> (1)

while on the second and the third pair it operates with op1 obtaining respectively:

<tidy-up, wash, eat | movable, horizontal, not-big> (2)
<relax, eat, wash | not-big, not-heavy, horizontal> (3)

These last objects (2) and (3), having a great deal of characteristics in common are associated once more with operation op2 obtaining the object:

<relax, tidy-up | movable, not-heavy> (4)

This object, which already belongs to a third level of complexity is associated with object (1) belonging to the second level of complexity.

Their association, through operation op3 gives the even more complex: object:

<relax, tidy-up, work | horizontal, not-heavy, movable, electronical> (5)

which is chosen by the Inverted Ockham's Razor Rule being the most complex generated object.

The test (to verify that it satisfies the user requests) is performed successfully on object (5) and later two predefined objects are individuated in it:

a (chair, <relax | not-heavy, horizontal>) and
a (robot, <work | movable, electronical>)

As they haven't any characteristics in commons they can be used to synthesize the new invented object which will then be defined as the following composition:

(chair, robot, <tidy-up> <movable>)

A best interpretation of this idea can be to associate a **chair** with a **robot** in order to obtain a chair which, after heving been used to **relax**, is able to come back to its nominal position.

The example shows that creativity of the invented object can be located in two different aspects. The former, necessarily maintaining the interactively requested structure, derives from the application of the association-rules. The latter, joining impredictable elements, emerges from the dissociation-rules.

4. ACKNOWLEDGEMENTS

We would like to tank Prof. Giovanni Degli Antoni for his fundamental contribution to the present work.

5. BIBLIOGRAPHY

[1] M. Boden, MECHANISMS OF CREATIVITY, Inaugural lecture, University of Sussex, 1982

[2] A. Bundy, L.S. Sterling, META-LEVEL INFERENCE IN ALGEBRA, Research paper 164, Dept of Artificial Intelligence, University of Edinburgh, Presented at the Workshop on Logic Programming for Intelligence Systems, Los Angeles, 1981

[3] A. Bundy, B. Welham, USING META-LEVEL INFERENCE FOR SELECTIVE APPLICATION OF MULTIPLE REWRITE RULES IN ALGEBRAIC MANIPULATION, Artificial Intelligence 16(2), 1981

[4] A. Bundy, METALEVEL INFERENCE AND

CONSCIOUSNESS, in The mind and the machine, by S. Torrance (ed.), (Ellis Horwood Series A.I.), 1984

[5] N. Chomsky, CURRENT ISSUES IN LINGUISTIC THEORY, The Hague, Mounton, 1966

[6] D.R. Hofstadter, CAN INSPIRATION BE MECHANISED?, in Scientific American, (pp. 18-27), September 1982

[7] D.R. Hofstadter, VARIATIONS ON A THEME AS AN ESSENCE OF IMAGINATION, in Scientific American, October 1982

[8] D.R. Hofstadter, VARIATIONS ON A THEME AS AN ESSENCE OF INSPIRATION, in Scientific American, (pp. 14-21), October 1982

[9] D.B. Lenat, THE UBIQUITY OF DISCOVERY, in Artificial Intelligence, 19 (pp. 257-285) 1978

[10] D.B. Lenat, ON AUTOMATED SCIENTIFIC THEORY FORMATION: A CASE STUDY USING THE A.M. PROGRAM, in (Hayes et al., eds.) Machine Intell. 9, N.Y.: Halstead Press, (pp. 251-283), 1979

[11] D.B. Lenat, THE NATURE OF HEURISTICS, in Artificial Intelligence 19 (pp. 189-249), 1982

[12] A. Newell, J.C. Shaw, H.A. Simon, THE PROCESSES OF CREATIVE THINKING, in H.E. Gruber, G. Terrel, M. Wertheimer (eds.) Contemporary Approaches to Creative Thinking, N.Y. Atherton Press (pp. 63-119), 1962

[13] A. Newell, H.A. Simon, HUMAN PROBLEM SOLVING, (Prentice Hall Englewood Cliffs, N.Y.), 1972

[14] G. Polya, HOW TO SOLVE IT, (Princeton University Press), 1945

[15] G. Polya, MATHEMATICS AND PLAUSIBLE REASONING, (Priceton University Press), 1954

[16] I. Prigogine, I. Stengers, LA NOUVELLE ALLIANCE, (Gallimard, Paris), 1979

[17] I. Prigogine, FROM BEING TO BECOMING - TIME AND COMPLEXITY IN PHYSICAL SCIENCES, (W.H. Freeman), 1980

[18] M. Yazdany, CREATIVITY IN MAN AND MACHINES, in The Mind and The Machine, by S. Torrance (ed.), (Ellis Horwood Series A.I.), 1984

THE INTERTEST INITIATIVE WITHIN THE OSIRIDE PROJECT

Luciano Lenzini

CNR-Istituto CNUCE - Pisa

RIASSUNTO

Il Consiglio Nazionale delle Ricerche (CNR), per realizzare la sua politica nel settore informatico, ha promosso nel 1982 il progetto OSIRIDE [1] con l'intenzione di offrire agli utenti della cominità scientifica italiana, che necessitano di cooperare in programmi di ricerca nazionali ed internazionali, un insieme uniforme di servizi informatici.

Tali servizi saranno forniti da computer di diversi costruttori disseminati sul territorio Italiano e cooperanti attraverso la rete di trasporto pubblica a commutazione di pacchetto X.25 ITAPAC, in accordo agli standard OSI (Open Sistems Interconnection) [2] e alle raccomandazioni CCITT. Attualmente sono stati coinvolti nel progetto OSIRIDE quattro costruttori (DIGITAL, Honeywell Bull, IBM, OLIVETTI), i quali, nella loro politica per gli sviluppi OSI, hanno già sviluppato prodotti che soddisfano le richieste di OSIRIDE.

Per verificare la interoperabilità dei prodotti OSI, sopra menzionati (da ora in poi chiamati OSI software), nel 1986, attraverso il progetto OSIRIDE, il CNR ha promosso un'iniziativa denominata OSIRIDE-Intertest. Si è conclusa (Giugno 1987) la prima fase dell'Intertest (Fase A), relativa alla verifica di interoperabilità dei protocolli dei primi cinque livelli.

Di seguiito sono stati evidenziati gli obiettivi di OSIRIDE, le scelte architetturali ed i ruoli dei partecipanti; analogamente é stato fatto per OSIRIDE-Intertest. È stata descritta in dettaglio la prima fase di OSIRIDE-Intertest ed evidenziati gli insegnamenti scaturiti.

I piani futuri e le conclusioni sono evidenziati alla fine.

Non si intende, con questo articolo, fornire elementi di comparazione sulla qualità del software fornito dai costruttori. Ogni costruttore ha incontrato problemi di qualità e severità diversi. Nel paragrafo 10 viene fornito un sommario contenente informazioni concrete sui risultati ottenuti. Tale lista riflette la globalità dell'esperienza OSIRIDE-Intertest, senza indicare la responsabilità individuale.

ABSTRACT

Promoted in 1982 by the Italian National Research Council (CNR) to implement its computing policy, the OSIRIDE project [1] is aimed at offering a uniform set of information services for Italian scientific users who need to cooperate in national and international research programs. These will be given to the scientific user community by the computers of different manufacturers spread out over Italy and cooperating through ITAPAC, the Italian X.25 public packet switching network, according to the Open Systems Interconnection (OSI) [2] Standards and CCITT Recommendations. At present, four manufacterers - DIGITAL, Honeywell-Bull, IBM and OLIVETTI - are involved in the OSIRIDE project and have already implemented products covering the OSIRIDE needs in their OSI development policy. At the beginning of 1986, in order verify the Interworking capabilities of the above-mentioned OSI products (from now on referred to as OSI software) CNR promoted, within the OSIRIDE project, an initiative named OSIRIDE-Intertest. The first phase of it (Phase A), which dealt with the verification of the interworking capabilities of the first five lower layers protocols was recently completed (June 1987).

In the following, the objectives of OSIRIDE and the related architectural choices are outlined. The OSIRIDE Partecipants and their related roles are then highlighted. The objectives and architecture of the verification system the OSIRIDE-Intertest Initiative deals with are described. The OSIRIDE-Intertest partecipants and their their related roles are also introduced. The first phase of OSIRIDE-Intertest is described in detail and lessons learned from it are then underlined. Future plans and conclusions are drawn at the end.

It is not intent of this paper to give any comparative assesment of software quality provided by the manufacterers. Differents problems were encountered. They were substantially different in quality and severity for each manufacterer.

In order to provide concrete information on the results obtained, a summary of the most significant errors is given in section 10 of the paper.

The list reflects the overall OSIRIDE-Intertest experience and no individual responsability is indicated.

1. INTRODUCTION

Since the second half of the 70's CNR's policy on the distribution of computung facilities among its institutes

has been based on the following choices:

— **Computeng Power Distribution**
The growing demand by national scientific users for computer facilities has given rise to the institution of CNR computing centers in areas where users exceed a given critical threshould. As a result of this choice there is now a great number of computing centers spread out all over Italy.

— **Hardware Heterogeneity**
CNR has maintained a position of equidistance in its relationship with computer manufacturers, encouragingheterogeneous choices for different geographical areas. Thus computers from several manufacturers are now operating in the above computing centers.

— **Networking**
Since it was difficult, if not impossible, for one single computing center to provide all the facilities needed by different fields of reasearch. CNR decided to integrate the facilities by interconnecting its own computing centers in a computer network. The interconnection of this computer network with other national and international (e.g. ARPANET, BITNET/EARN, etc.) computer networks was also forseen.

Right from the beginning of 1979, CNR started experimenting this policy by initially using RPCNET: a packet switching distributed computer network designd and developed in the 70's by CNECE in collaboration with other scientific partners. However RPCNET was not adeguate enough to fully implement the policy previously sketched, as:

— RPCNET was the result of a research project among research institutions and was not intended to be used in an operational environment;

— RPCNET was implemented only on IBM machines running VM/370 due to skill and the limited manpower available;

— RPCNET protocols did not comply (as it was designed in the first half of the 70's) with the new emerging OSI Standards and CCITT Recommendations.

In order to get rid of these limitations in the first half of 1982 CNR promoted a project called OSIRIDE (**O**pen **S**ystems **I**nterconnection su **R**ete **I**taliana **D**ati **E**terogenea - OSI an Heterogeneous Italian Data Network) which was conceived at CNUCE (a CNR Institute lacated in Pisa) by a team of people (Known as **"OSIRIDE team"**).

The following partners are now cooperating with CNR in the OSIRIDE project:

— **DIGITAL, Honeywell Bull, IBM** and **OLIVETTI (from now on referred to as OSIRIDE Vendors);**

— **a software development company (TECSIEL);**

— the branch (**SIP**) of the Italian PTT which is in charge of managing ITAPAC.

However, OSIRIDE is an open project as it can be extended to other partners, according to modalities already defined by CNR.

2. OSIRIDE OBJECTIVES AND CHOICES

In order to fully implement the CNR policy previously outlined the OSIRIDE team made the following choices:

— Adoption of the OSI/RM (IS 7498).

— Adoption (Table 1) of the following OSI Standards and CCITT Recommedations for each of the seven layers:

Physical	**V.24**
Data Link	**HDLC (LAPB)**
Network	**X.25 (1980)**
Transport	**ISO 8072, 8073 (Classes 0,2)**
Session	**ISO 8326, 8327 (Full Functionally)**
Presentation	**ISO 8822, 8823 (Full Functionally)**
Application	**FTAM, MHS**

Table 1: Initial OSIRIDE Choices.

The rationale behind these choices can be summarized as follows (Figure 1):

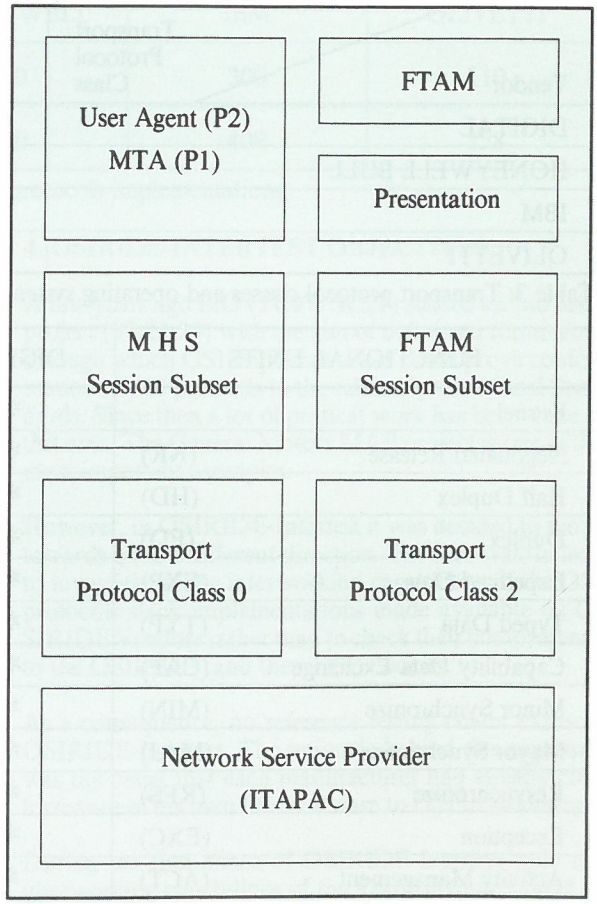


Figure 1 OSIRIDE Protocol layering.

- V.24, HDLC (LAPB) and X.25 (1980) were requested because of ITAPAC.
- Transport Protocol classes 0 and 2 were requested because of ITAPAC and due to the fact that MHS and FTAM were recognized to be the most urgent application standards for the CNR scientific community.
- The full Session Layer functionality was requested because of FTAM and MHS.
- The full Presentation Layer functionally was requested because of FTAM.

— Design of a Management System for managing and coordinating the global OSIRIDE system to allow for political, administrative and technical decisions. As no DIS was (and is not yet) available in the Management area the OSIRIDE team designd a Management System with the idea of converging towards OSI as soon as the related OSI Standard would be available.

— Selection of several computing centers - named **OSIRIDE Pilot Centers** - hosting **DIGITAL, Honeywell Bull, IBM** and **OLIVETTI** computers used as guinea pigs for experimenting OSI software of the OSIRIDE project.

— Interconnection of the OSIRIDE Pilot Centers by means of ITAPAC.

AS OSIRIDE is up to date on the evolution of OSI Standards and CCITT Recommendations, the OSIRIDE team has already forseen a set (Table 2) of technologies (in addition to ITAPAC) and Application Protocols (besides FTAM and MHS) that could be medium-term candidates for the OSIRIDE project.

— **Ethernet and Token Ring LANs**
— **Distributed Directory Service**
— **CCR (Commitment, Concurrency and Recovery)**
— **Transaction Processing**
— **GKS (Graphical Kernel System)**

Table 2: Future OSIRIDE Extensions.

It must be observed that the adoption of LAN technologies will very likely imply the adoption of class 4 for the Transport Layer protocol.

3. OSI MANUFACTERERS SOFTWARE CHARACTERISTICS

All the OSIRIDE Vendor operating syxtems have been supporting the X.25, conforming to ITAPAC, for some time.

Vendor \ Transport Protocol Class	0	1	2	3	4	Operating Systems
DIGITAL	X		X		X	VMS
HONEYWELL BULL			X		X	GCOS6
IBM	X		X			VM/SP4
OLIVETTI	X		X	X	X	MOS L1

Table 3: Transport protocol classes and operating systems.

FUNCTIONAL UNITS		DIGITAL	HONEYWELL	IBM	OLIVETTI
Kernel		*	*	*	*
Negotiated Release	(NR)	*	*		*
Half Duplex	(HD)	*	*	*	*
Duplex	(FD)	*	*	*	*
Expedited Data	(EXP)	*	*		*
Typed Data	(TYP)	*	*	*	*
Capability Data Exchange	(CAP)	*			
Minor Synchronize	(MIN)	*	*	*	*
Mayor Synchronize	(MAJ)	*	*	*	*
Resynchronize	(RES)	*	*	*	*
Exception	(EXC)	*	*	*	*
Activity Management	(ACT)	*		*	*

Table: 4 Session Functional Units.

QUALITY OF SERVICE PARAMETERS	DIGITAL	HONEYWELL	IBM	OLIVETTI
Extended Control	*	*		*
Oprimized Control		* □		

* = yes — □ = able to manage only on receipt.

Table 5: Q.O.S. parameters.

DIGITAL	HONEYWELL	IBM	OLIVETTI
65535	32767	999999	6144

Table 6: Maximum lenght (in bytes) of NSSDUs implemented by the OSIRIDE vendors.

DIGITAL	HONEYWELL	IBM	OLIVETTI
		*	*

Table 7: SSDU segmentation.

	DIGITAL	HONEYWELL	IBM	OLIVETTI
Transport	96	120	300	110
Session	82	90	408	158

Table 8: Object module sizes (in Kbytes) for Transport ans Session protocols implementations.

Tables 3, 4, 5, 6 and 7 make up a summary of the Transport Protocol Classes, Session Functional Units, Quality of Service Parameters, Normal Session Service Data Unit (NSSDUs) maximum lenght and segmentation capabilities implemented by the OSIRIDE Vendors in theri OSI software and made available to the OSIRIDE environment. Table 3 also shows the operating systems running in the OSIRIDE Pilot Centers. Table 8 shows the object module sizes for Transport and Session protocol implementations related to the OSIRIDE Vendors. It can be seen from Tables 3 and 4 that the choices so far performed by the OSIRIDE Vendors for the Transport and Session Layers reflect OSIRIDE's priorities quite well.

Since 1985 the OSI software implementing the first lower five layers protocols for OSI, made available by the OSIRIDE Vendors, has been successfully verified among Pilot Centers hosting homogeneous computers.

The next step forward was then to verify the interoperability among heterogeneous computers. This is the subject of an Initiative promoted by CNR at the beginning of 1986 and lead by CNUCE (still in the framework of the OSIRIDE project) named **OSIRIDE-Intertest.**

4. OSIRIDE-INTERTEST OBJECTIVES

A few years ago ISO (TC97/SC21) started an "ad hoc" project (97.21.23) with the aim of defining a framework through which OSI implementators could prove conformance of OSI products to the related International Standards. Since then a lot of pratical work has been done in this area. The General Motors MAP project is one of the most important examples.

However, in OSIRIDE-Intertest it was decided to move towards quite a different direction. The idea was, in fact, to investigative the interworking capabilities of the OSI protocols stack implementations made available by O-SIRIDE Vendors rather than to check their conformance to the OSIRIDE (and then OSI) choices.

As a consequence, no reference testing center exists in OSIRIDE-Intertest. The retionale behind this approach was the belief that each manufacturer had already conformance of his own OSI software to OSI in laboratory.

During the first phase of OSIRIDE-Intertest only the interworking capabilities of the OSI software related to the first five lower layers - from now on referred to as

OSI/5 packages - have been successfully verified. This allows the OSIRIDE environment (Made up of Pilot Centers connected to ITAPAC) to be ready to verify the interoperability of the Presentation Layer Protocol, FTAM and MHS, without being concerned with the debugging of the debugging of the protocols underneath.

Additional objectives of the OSIRIDE-Intertest Initiative that CNR is determined to achieve are:

— the strengthening of CNR/Industry relationships which have already producedvery important results for CNR;

— the establishment of technical/organizational know-how on interoperability.

5. OSIRIDE-INTERTEST PARTECIPANTS AND RELATED ROLES

The OSIRIDE-Intertest Initiative Partecipants are the same ones as those involved in the OSIRIDE project; i.e. CNR, Vendors, TECSIEL and SIP. Again, OSIRIDE-Intertest is also an open initiative as it can be exetended to other Vendors, according to modalities already defined by CNR.

In order to carry on their activities, the organizations involved in OSIRIDE-Intertest have representatives in a steering committee - chaired by CNR - which meets periodically to:

— define and plan the activitied needed, in order to achieve the objectives set by CNR for OSIRIDE-Intertest;

— evaluate the progress of the activities;

— examine the problems that could arise during the execution of the initiative;

— arrange for the publication of the information.

The roles played by the OSIRIDE-Intertest partners can be summarized as follows:

CNR

— leads OSIRIDE-Intertest;

— provides the design of the OSIRIDE-Intertest Architecture, the related implementation specs and the methodology necessary for verifying interworking;

— defines the test sequences (see below);

— equipes Pilot Centers with operating systems adequate for hosting the OSI software implemented by OSIRIDE Vendors;

— organizes the domos in agreement with the other partners;

— advertises, in agreement with the other partners, the relevant events of OSIRIDE-Intertest;

— manages partner "liasons".

The last point is one of the most delicate and important ones. Manufactures can never work tegether in OSIRIDE-Intertest without the presence of CNR which has the role of cooperating with them and managing their liasons. This is role fundamental, for example, when manufacturers give different interpretations af a specific aspect of an OSI protocol or a CCITT Recommendation.

OSIRIDE Vendors

— provide OSI software to CNR;

— provide the development of the Scenario Interpreter (see section 8);

— carry out the verification activity at Pilot Centers.

TECSIEL

TECSIEL collaborates with CNR in the executive management of OSIRIDE-Intertest. That is, it:

— provides assistance to CNR and through CNR provides assistance to the Vendors during the execution of OSIRIDE-Intertest at Pilot Centers;

— develops the software components of the OSIRIDE-Intertest architecture which are common to all the Vendors (see section 8);

— produces the scenario sequences defined by CNR and elaborates the test results in order produce readable reports.

SIP

— assists CNR in all the aspects dealing with the interconnection of Pilot Centers to ITAPAC;

— provides CNR, on request, with the so-called ITAPAC Trace (taken at the packet switching nodes) through which it is possible to establish the neutral bit configuration of the various packets exchanged between Pilot Centers.

In the following, Phase A of OSIRIDE-Intertest is described in detail.

6. OSIRIDE-INTERTEST CHARACTERISTICS

Within Phase A the main goals of the OSIRIDE-Intertest Initiative was the design and implementation of an architecture [3] - named **OSIRIDE-Intertest architecture** - suitable for checking the interworking capabilities between the OSI/5 packages running at the OSIRIDE Pilot Centers.

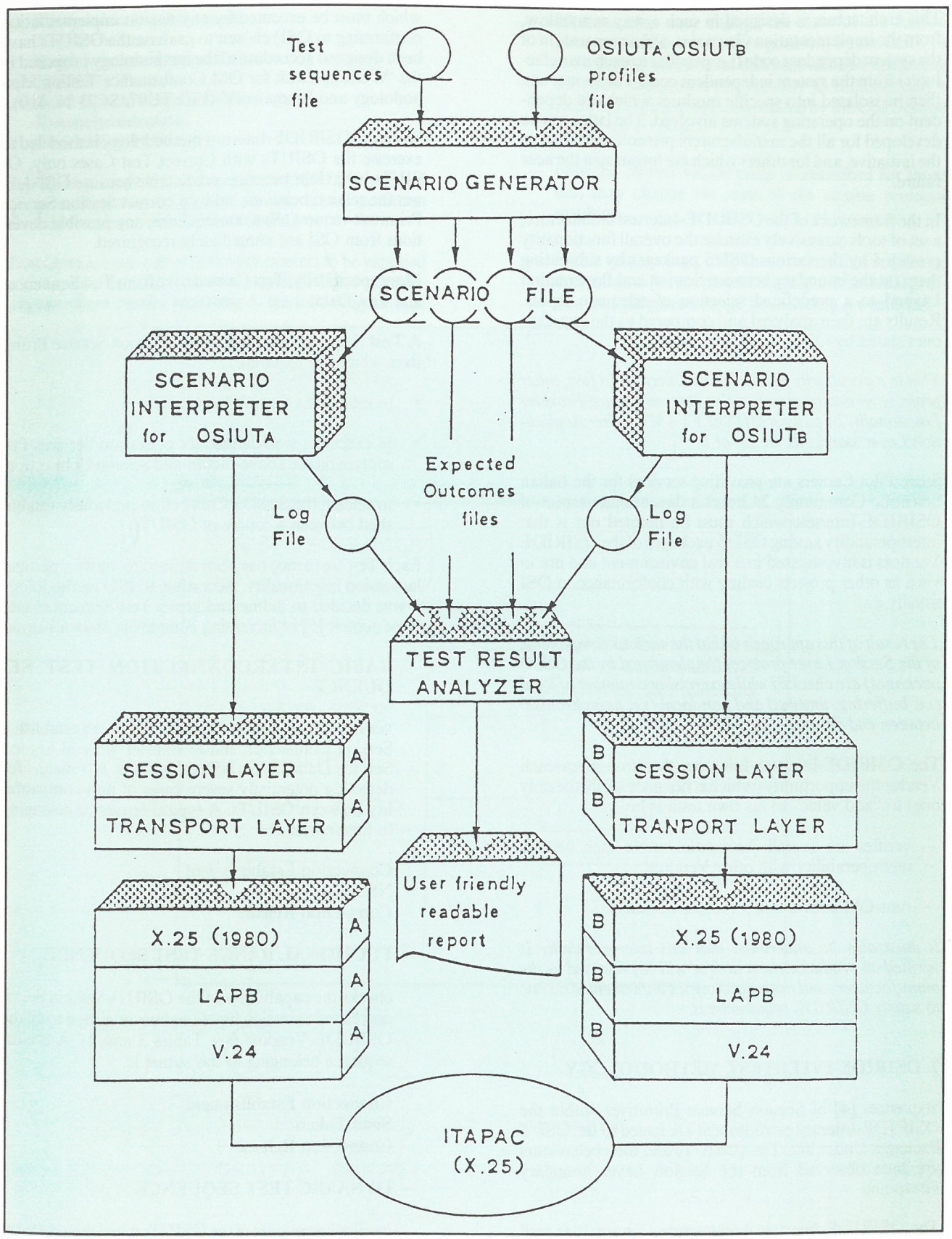


Figure 2 OSIRIDE Intertest architecture.

This architecture is designed in such a way as to allow, from the implementation viewpoint, a clear separation of the system dependent code (i.e. peculiar to each manufacturer) from the system independent code. The former can then be isolated into specific modules which are dependent on the operating systems involved. The latter can be developed for all the manufacturers presently involved in the initiative, and for others which are foreseen in the near future.

In the framework of the OSIRIDE-Intertest architecture, a set of tools extensively exercise the overall functionality provided by the various OSI/5 packages by submitting them (at the boundary between Session and Presentation Layers) to a predefined selection of adequate inputs. Results are then analyzed and compared to the expected ones stated by OSI.

This is a peculiarity of OSIRIDE-Intertest. In fact, other projects vertically exercises the Session Service Provider functionality by putting MHS or FTAM or other Application Layer standards on top of it.

Since Pilot Centers are providing services for the Italian Scientific Community 24 hours a day, another aspect of OSIRIDE-Intertest which must be pointed out is that interoperability among OSI/5 packages of the OSIRIDE Vendors is investigated in a real environment and not in vitro as other projects dealing with conformance to OSI usually do.

The result of this approach is that the various components of the Session Layer protocol (implemented by the OSI/5 packages) are checked while exercising a number of local (i.e. buffering, timeout) and non-local (i.e. asynchronism between end-systems) factors.

The OSIRIDE-Intertest Initiative therefore gives each Vendor the opportunity (which is not necessarily the only one) to "add value" to his own tests as he:

— verifies the overall (horizontal rather then verticla interoperability with other Vendors;

— runs OSI software in a real environment.

It must also be underlined that this interoperability is verified on products which are (or will be) released by the manufacturers and not on prototypes implemented ad hoc to satisfy OSIRIDE requirements.

7. OSIRIDE-INTERTEST METHODOLOGY

Sequences [4] of Session Service Primitives within the OSIRIDE-Intertest environment are issued to the OSI/5 Packages Under interTest (OSIUT) and their behaviours are then observed from the Session Layer boundary viewpoint.

The OSIRIDE-Intertest Architecture (Figure 2) as well as Test Cases (i.e. sequences of Session Service Primitives which must be executed by any session implementation conforming to OSI) chosen to exercise the OSIUTs have been designed according to the methodology proposed in the 'Working Draft for OSI Conformance Testing Methodology and Framework' (ISO TC97/SC21 N. 410).

Since the OSIRIDE-Intertest methodology is modeled to exercise the OSIUTs with Correct Test Cases only, OSIUTs behaviour becomes predictable because OSI defines the session behavior led by a correct Session Service Primitive request. As a consequence, any possible deviations from OSI are immediately recognized.

More specifically, Test Cases derive from Test Sequences and Test Data.

A **Test Sequence** is a sequence of Session Service Primitives which are required:

- to establish a Session Connection;
- to execute a well-defined set of Session Services Primitives on the above-mentioned Session Connection;
- to release the Session Connection previously established between a couple of OSIUTs.

Each Test Sequence has been defined to verify a particular session functionality. According to ISO methodology it was decided to define and group Test Sequences into three subtest [5] of increasing complexity shown below:

— **BASIC INTERCONNECTION TEST SEQUENCE**

verifies the capabilities of the OSIUTs to establish a Session Connection and to transfer Normal Session Service Data Units (**NSSDUs**). This is a means for detecting potentially severe cases of non-compatibility between OSIUTs. A typical sequence belonging to this subtest is:

Connection Establishment
NSSDUs Transfer
Connection Release

— **FUCTIONAL RANGE TEST SEQUENCE**

checks the capabilities of the OSIUTs Session Services related to session functionalities common to all the OSIRIDE Vendors (see Tables 3 and 4). A typical sequence belonging to this subset is:

Connection Establishment
Send Token
Connection Release

— **DYNAMIC TEST SEQUENCE**

verifies behaviours of the OSIUTs when they perform more complex instances of communication, as:

Connection Establishment
NSDDUs Transfer
Send Synchronization Point
NSDDUs Transfer
Send Synchronization Point again
Resynchronization
Connection Release

During this phase each couple of OSIUTs is stressed in such a way as to test all the related common functionalities (see Tables 3 and 4).

Test Data are the values of the parameters to be supplied with Session Services Primitive requests. These parameters have been chosen according to the following criteria:

— all those that may change the state of the session protocol machine (i.e. token position) by the OSIUTs are supplied with several different values. Concerning the other parameters (i.e. Activity Identifiers), only a value belonging to the range stated by the OSI Session Standards is chosen;

— only the correct values range is examined for those that may change the state of the session protocol machine;

— in this values range, only those values that require or may cause different processing (e.g. token initial position and parameter range boundaries) are selected.

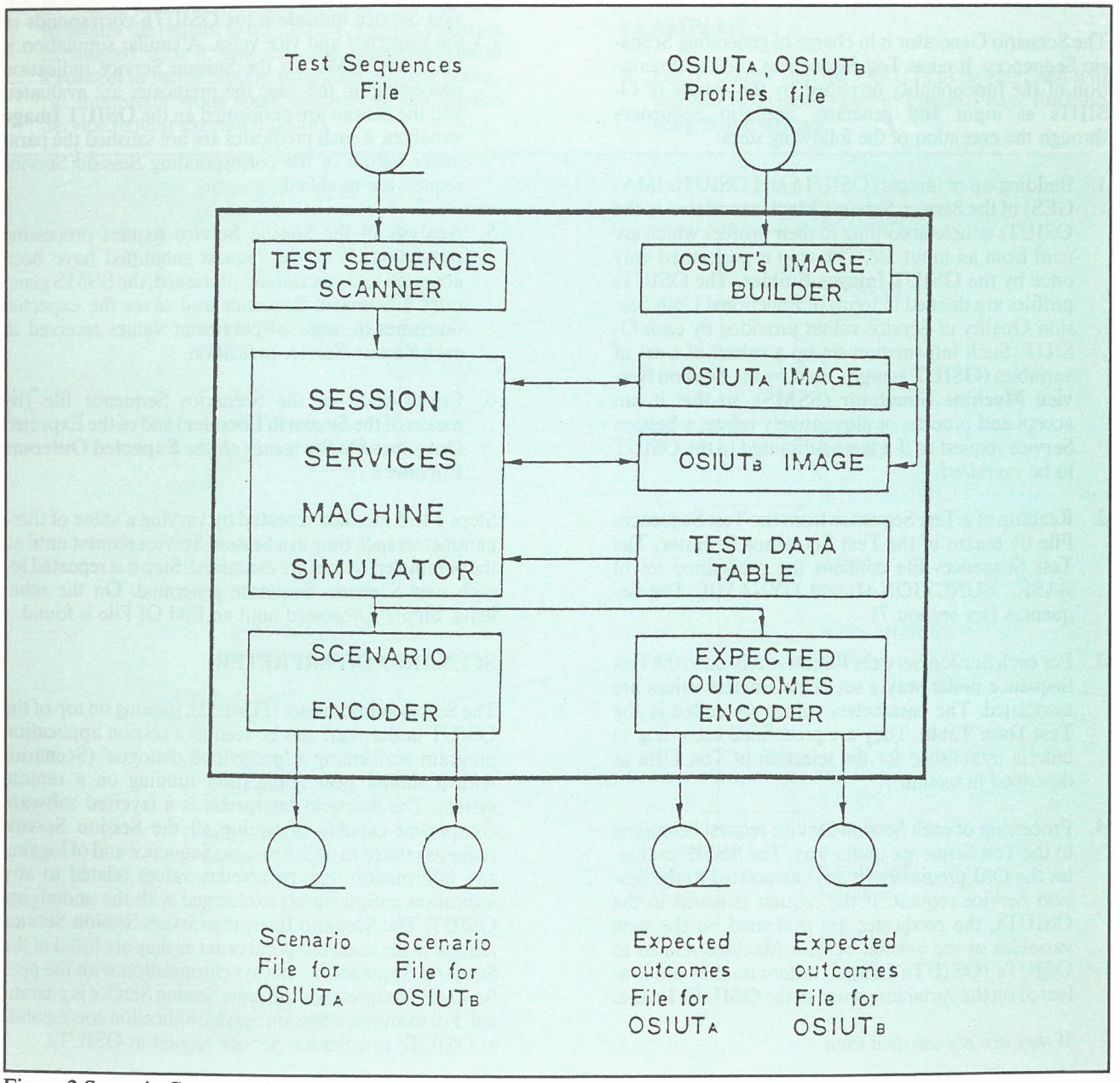


Figure 3 Scenario Generator Structure.

A correct combination of Test Data is related to each Test Sequence and the resulting sequence is a Test Case also called Scenario Sequence.

Up to 350 Scenario Sequences have been generated from the predefined Test Sequences for a couple of OSIUTs encompassing the whole OSI functionality ('full session').

8. OSIRIDE-INTERTEST ARCHITECTURE

The OSIRIDE-Intertest Architecture is structured into two major components: the Scenario Generator (Figure 3) and the Scenario Interpreter (Figure 2).

SCENARIO GENERATOR

The Scenario Generator is in charge of generating Scenario Sequences. It takes Test SEquences and the description of the functionality provided by the couple of O-SIUTs as input and generates Scenario Sequences through the execution of the following steps:

1. Building up of images (OSIUTa and OSIUTb IMAGES) of the Session Services Machines related to the OSIUTs couple according to their profiles which are read from an input file. This step is performed only once by the **OSIUT Images Builder.** The OSIUTs profiles are defined in terms of Functional Units Session Quality of Service values provided by each O-SIUT. Such information set-up a subset of a set of variables (**OSIUT Image**), used by the **Session Service Machine Simulator (SSMS),** so that it can accept and process, or alternatively refuse, a Session Service request as if it were submitted to the OSIUT to be exercized.

2. Reading of a Test Sequence from the Test Sequences File by means of the **Test Sequence Scanner.** The Test Sequences file contains the predefined set of BASIC, FUNCTIONAL and DYNAMIC Test Sequences (see section 7).

3. For each Session Services Primitive related to the Test Sequence under way a set of parameters values are associated. The parameters values are coded in the **Test Data Table.** They are predefined according to criteria established for the selection of Test Data as described in section 7.

4. Processing of each Session Service request belonging to the Test Sequence under way. The **SSMS** evaluates the OSI predicates (if any) associated to the Session Service request. If the request is issued to the OSIUTa, the predicates are evaluated on the state variables of the Session Service Machine related to OSIUTa (**OSIUTa Image**), otherwise they are evaluated on the variables related to the **OSIUTb Image.**

   If *they are not satisfied* **then**

   the SSMS changes the value of some parameters in the Session Service request and evaluates the predicates again.
   (This step is repeated until either the predicates are satisfied or all the parameter values have been examined for the current Session Service request and for all the previous ones. In the latter case the generation is stopped and the control returns to step 2).

   else

   the **SSMS** simulates the processing of the Session Service request by performing the actions on the **OSIUTa image** variables the actions related by OSI Standards to the request.

   To each Session Service request for OSIUTa, a Session Service indication for OSIUTb corresponds in the Sequence and vice versa. A similar simulation is made by **SSMS** for the Session Service indication processing: in this case the predicates are evaluated and the actions are performed an the **OSIUT Image** variables; if such predicates are not satisfied the parameter values of the corresponding Session Service request are modified.

5. Analysis of the Session Service request processing simulation. If all the request submitted have been accepted and successfully processed, the **SSMS** generates a Scenario Sequence and saves the expected outcomes in terms of parameter values received in each Session Service indication.

6. Construction of the Scenarios Sequence file (by means of the **Scenario Encoder**) and of the Expected Outcomes file (by means oh the **Expected Outcome Encoder**).

Steps 3 to 5 are then repeated by varying a value of these parameters each time in a Session Service request until all the parameters values are examined. Step 6 is repeated for each new Scenario Sequence generated. On the other hand, Step 2 is repeated until an End Of File is found.

SCENARIO INTERPRETER

The Scenario Interpreter (Figure 2), running on top of the OSIUT under way, can be seen as a session application program performing a 'predefined dialogue' (Scenario) with a similar peer application running on a remote system. The Scenario Interpreter is a layerred software component capable of issuing all the Session Service requests related to each Scenario Sequence and of logging any information (e.g. parameters values related to any indication, return status) exchanged with the underlying OSIUT. The Scenario Interpreter issues Session Service request in the same temporal order as they are listed in the Scenario Sequence, so that synchronization with the peer Scenario Interpreter on the same Session Service is guaranted. For example, a Session Service indication corresponds at OSIUTb to a Session Service request at OSIUTa.

The Scenario Interpreter functions are grouped in the

following components:

— An **Interface** to the OSIUT. This component provides a mapping between Session Service Primitives and software macros (peculiar to each Vendor) through which the proper OSIUT functionality is invoked.

— A **Synchronization Protocol** which maintains synchronization with the peer Interpretes on the same primitive of the Scenario Sequence under way.

— A **Scanner** of Scenario Sequences stored in the Scenario Sequence file.

— A **Driver** which (for each Session Service request belonging to a Scenario Sequence) calls the **Interface** to submit the request to the OSIUT, logs the messages exchanged with it, and uses the **Synchronization Protocol** to resynchronize itself with the peer Scenario Interpreter when some anomaly in the Session Service request execution occurs.

9. RESULTS ANALYSIS

A results analysis is made on completion of each Scenario Sequences file processing. Since the Scenario Sequence submitted to the PSIUT is correct (i.e. the OSIUTs should be able to execute all the Session Service request belonging to the Scenario), any exchange of SSDUs between the OSIUTs is predetermined.

An analysis of the results is made by an off-line tool named **Test Result Analyzer** which rebuilds the sequences of SSDUs and compares them with expected ones. A report listing the results of the Scenario Sequences processing is written in a user friendly format. It also underlines all the misbehaviours detected.

EXAMPLES

1. Let us assume that the OSIUTs to be exercised with the Test Sequence drawn in Figure 4 have the following characteristics:

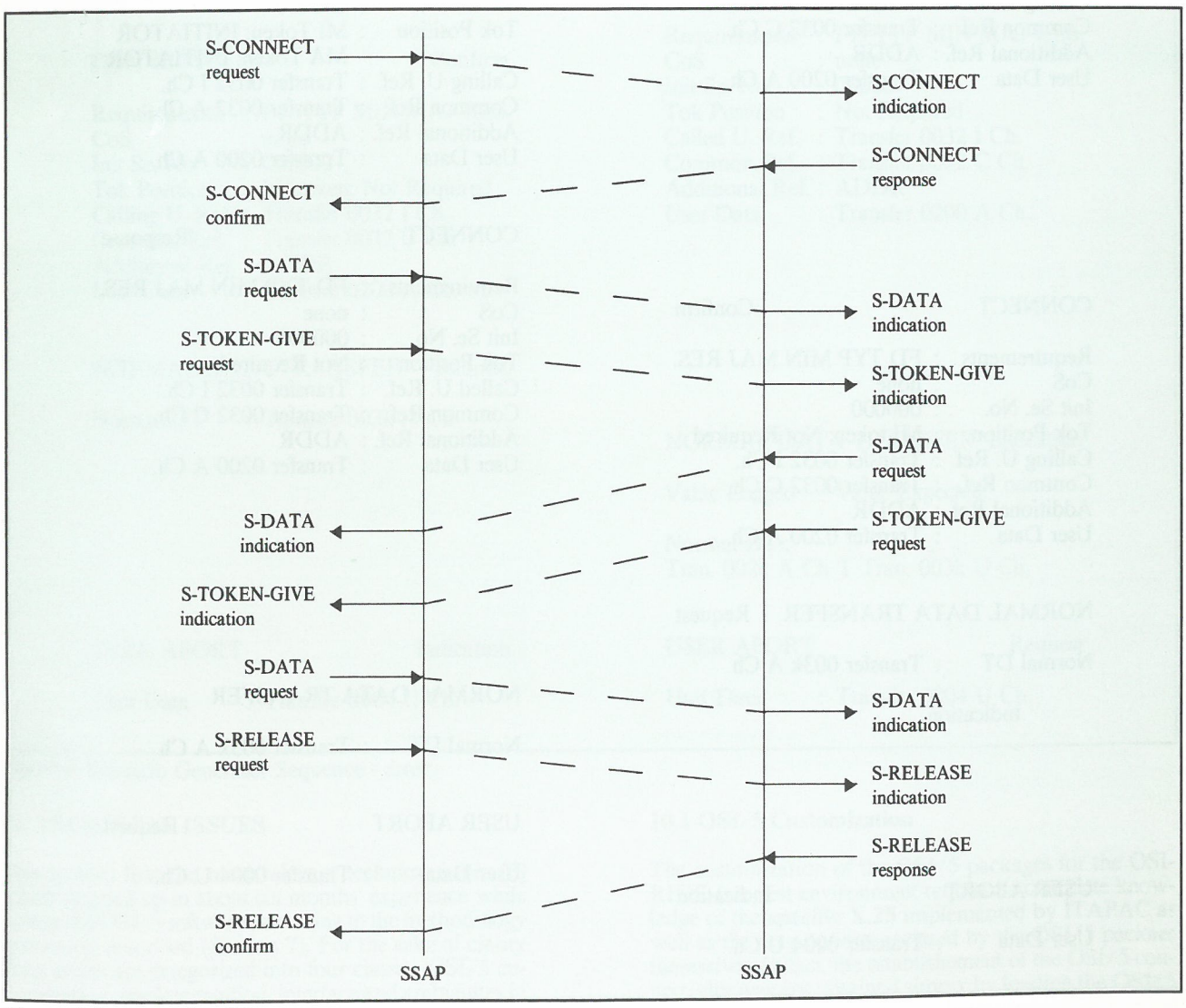


Figure 4 An example of Test sequence.

— OSIUTa supports Full Duplex, Typed Data, Minor and Major Synchronize and Resynchronize Functional Units;

— OSIUTb implements a full session.

All the inputs needen by the Scenario Generator are supplied by enconding such information in the *OSIUTs profile* file and by recording the Test Sequence in the *Test Sequence* file. A particular Scenario Sequence generated by the Scenario Generator is shown in Figure 5.

2. If one of the OSIUTs does not work correctly, the Test Result Analyzer points out the anomaly detected in the manner shown in Figere 6. In the example, an erreor is detected by OSIUTb which receives Normal DATA of lenght 2K characters rather than 3K as expected in the NORMAL DATA INDICATION.

SESSION IMPL. A SEQ.__NUM. 0000

CONNECT Request

Requirements : FD TYP MIN MAJ RES
CoS : none
Init Se. No. : 000000
Tok Position : MI Token: INITIATOR
MA Token: INITIATOR
Calling U. Ref. : Transfer 0032 I Ch.
Common Ref. : Transfer 0032 C Ch.
Additional Ref. : ADDR
User Data : Transfer 0200 A Ch.

CONNECT Confirm

Requirements : FD TYP MIN MAJ RES.
CoS : none
Init Se. No. : 000000
Tok Position : MI token: Not Required
Calling U. Ref. : Transfer 0032 I Ch.
Common Ref. : Transfer 0032 C Ch.
Additional Ref. : ADDR
User Data : Transfer 0200 A Ch.

NORMAL DATA TRANSFER Request

Normal DT : Transfer 003k A Ch

Indication

USER ABORT Indication

User Data : Transfer 0004 U Ch.

SESSION IMPL. B SEQ.__NUM. 0000

CONNECT Indication

Requirements : FD TYP MIN MAJ RES
CoS : none
Init Se. No. : 000000
Tok Position : MI Token: INITIATOR
MA Token: INITIATOR
Calling U. Ref. : Transfer 0032 I Ch.
Common Ref. : Transfer 0032 A Ch.
Additional Ref. : ADDR
User Data : Transfer 0200 A Ch.

CONNECT Response

Requirements : FD TYP MIN MAJ RES.
CoS : none
Init Se. No. : 000000
Tok Position : Not Required
Called U. Ref. : Transfer 0032 I Ch.
Common Ref. : Transfer 0032 C Ch.
Additional Ref. : ADDR
User Data : Transfer 0200 A Ch.

NORMAL DATA TRANSFER

Normal DT : Transfer 003k A Ch

USER ABORT Request

User Data : Transfer 0004 U Ch.

Figure 5 Scenario Generator Sequence - no error.

SESSION IMPL. A SEQ.__NUM. 0000

CONNECT Request

Requirements : FD TYP MIN MAJ RES
CoS : none
Init Se. No. : 000000
Tok Position : MI Token: INITIATOR
MA Token: INITIATOR
Calling U. Ref. : Transfer 0032 I Ch.
Common Ref. : Transfer 0032 C Ch.
Additional Ref. : ADDR
User Data : Transfer 0200 A Ch.

CONNECT Confirm

Requirements : FD TYP MIN MAJ RES.
CoS : none
Init Se. No. : 000000
Tok Position : MI token: Not Required
Calling U. Ref. : Transfer 0032 I Ch.
Common Ref. : Transfer 0032 C Ch.
Additional Ref. : ADDR
User Data : Transfer 0200 A Ch.

NORMAL DATA TRANSFER Request

Normal DT : Transfer 003k A Ch

USER ABORT Indication

User Data : Transfer 0004 U Ch.

SESSION IMPL. B SEQ.__NUM. 0000

CONNECT Indication

Requirements : FD TYP MIN MAJ RES
CoS : none
Init Se. No. : 000000
Tok Position : MI Token: INITIATOR
MA Token: INITIATOR
Calling U. Ref. : Transfer 0032 I Ch.
Common Ref. : Transfer 0032 A Ch.
Additional Ref. : ADDR
User Data : Transfer 0200 A Ch.

CONNECT Response

Requirements : FD TYP MIN MAJ RES.
CoS : none
Init Se. No. : 000000
Tok Position : Not Required
Called U. Ref. : Transfer 0032 I Ch.
Common Ref. : Transfer 0032 C Ch.
Additional Ref. : ADDR
User Data : Transfer 0200 A Ch.

NORMAL DATA TRANSFER Indication

Value Logged Value Expected

Normal DT :
Tran. 002k A Ch I Tran. 003k U Ch.

USER ABORT Request

User Data : Transfer 0004 U Ch.

Figure 6 Scenario Generator Sequence - error.

10. TECHNICAL ISSUES

This section lists the most relevant technical issues [6] which popped up in about six months' experience while runnig the OSI/5 software according to the methodology previously described (Section 7). For the sake of clarity these issues are categorized into four classes: OSI/5 customization, implementation, interface and ambiguites in the standards.

10.1 OSI/5 Customization

The customization of the OSI/5 packages for the OSI-RIDE-Intertest environment required a complete knowledge of the **specific X.25** implemented by ITAPAC as well as the way options are used by the OSI/5 packaes themselves. In fact, the establishement of the OSI/5 connectivity was not obtained simply by loading the OSI/5 packages and starting them up. It was rather the result of

an intense cooperation among manufacterers and SIP.

10.2 Implementation

As expected (Section 6.), timing problems stemming from the real user environment in which the OSI/5 packages were running triggered a number of bugs in the session and transport protocols implementations. Weeding out these bugs was really a very patient trimming process involving all the OSIRIDE-Intertest partners as well as the manufacturer development laboratories. However it is to be observed that the number of bugs which came out during the OSIRIDE-Intertest runs is different for each manufacturer.

Session Protocol

— no signalling of a protocol error when the remote Session Entity sends a wrong SPDU not containing the Initial Serial Number (ISN) when it is requested (connect session service primitive);

— no recognition of default tokens by the remote Session Entity (S-Connect.Indication);

— error in coding the **token item** field (S-CONNECT. Response SPDU);

— error on collisione of Activity primitives;

— sending of a wrong Minor Sync Response PDU.

Transport Protocol

— no correct handling of the ACK TPDUs window;

— loss of an incoming CR TPDU;

— rejection of opening a Transport Connection immediately after system startup.

10.3 Session Interface

Some OSI/5 interfaces do not return to its SS-Users (Scenario Interpreters):

- some QoS parameter values;
- full information on token position;
- some Functional Units added by the **acceptor** in the opening phase of a Session Connection and not supported by the **initiator;**
- the serial number in the S-MINOR/MAJOR.Request and in the S-ACTIVITY-END.Request primitives.

Stepping trough the above points the conclusion is that, so far as the SS-User is concerned, he cannot insulate himself completely from the specific OSI/5 session interface. If service conformance were enforced in addition to protocol conformance the above problems would not exist.

10.4 Session Standard

— **Error**

In the case of rejection of a Session Connection the **reason code field** of the REFUSE SPDU is set to the value zero, which means **not specified.** On the other hand, in the related Session Service Primitive (S-CONNECT.Confirm) the **result parameter** can assume 2 values - **reject by SS-User** and **reject By SS-Provider** both with unspecified reason.

— **Ambiguity in the use of the Transport Expedited Data flow**

An ABORT SPDU must be sent on the Transport Expedited Data flow if the service was agreed during Transport Connection opening, otherwise this must be sent on Normal Data flow. It may also happen that an ABORT SPDU is to be sent during the Session Connection establishment phase. In this case, it is not clear from the Session Protocol whether the SPDU has be sent or not on the Transport Expedited Data flow, when this is available. It has been noted that in some OSI/5 packages, during the Session Connection establishment phase, the ABORT SPDU is sent on the Expedited Data flow, and on the Normal Data flow in other OSI/5 packages. This behaviour does not however change the interworkability between the computers, which is still fully guaranteed.

CONCLUSIONS

The OSIRIDE-Intertest Architecture implementation has been available to all the four Manufacturers involved in OSIRIDE-Intertest since December 1986. Intertest activities at the OSIRIDE Pilot Centers began on the 19th of January 1987 and were completed in June 1987. As expected, the execution of test sequences in a real environment has been very useful.

The Scenario Generator and the Test Result Analyzer implemented by TECSIEL are available on a VAX machine running VSM. Their object module sizes (excluding any library moduls) are summarized in Table 9. Table 10 summarizes the Scenario Interpreter object size related to the OSIRIDE Vendors.

Tool	Size
Scenario Generator	75
Test Result Analyzer	80

Table 9 Object module sizes of the Common Tools (in Kbytes).

	DIGITAL	HONEYWELL	IBM	OLIVETTI
Scenario Interpreter	100	120	20	117

Table 10 Scenario interpreter object module size (in Kbytes).

Phase A of OSIRIDE-Intertest was considered extremely positive by all the involved partners. Bugs were found in the OSI/5 products and promptly fixed by manufacturers, but more important than that, a number of ambiguities were pointed out in the present version of the ISO documents dealing with the Session and Transport Layers. Defect reports were formulated for examination within the related ISO Working Groups.

The best proof of the success of OSIRIDE-Intertest is that discussions on starting Phase B have already begun. Its objective is to verify the interworking capabilities of MHS according to the principle underlined in section 6.

ACKNOWLEDGEMENTS

Many persons from all the partners involved in OSIRIDE contributed significantly to the ideas and results presented in this paper.

Reaching cooperation among multiple competitive organizations was certainly a remarkable achievement in **human protocol** obtained within OSIRIDE. This has been possible thanks to the particular environment created by the **Commissione Generale per l'Informatica del CNR** to which all the partners involved in OSIRIDE wants to express their gratefulness.

C. Martinelli, A. Pezzulli and F. Zoccolini from TECSIEL contributed many helpful suggestions to the presentation and content of the paper, for which the author is very grateful.

REFERENCES

[1] F. Caneschi, E. Gregori, L. Lenzini, OSIRIDE: State of the Art and Future Developments, Proceeding of the Eighth International Conference on Computer Communication, Munich, F.R.G., 15-19 September 1986, pp. 84-88

[2] ISO 7498, Information Processing systems - Open Systems Interconnection - Basic Reference Model.

[3] OSIRIDE Multivendor Interworking Testing - Technical Specifications, Document No 6, 30/August/1986

[4] OSIRIDE Multivendor Interworking Testing, Test Sequences and Test Data, Document No 4, 11/June/1986

[5] OSIRIDE Multivendor Interworking Testing, General Description, Document No 2, 24/March/1986

[6] OSIRIDE Multivendor Interworking Testing, Final Report-Document No 11, 21/Dec./1987 (Confidential).

IL NOTIZIARIO SUI CD-ROM

NOVITÀ DA SEATTLE - *a cura di Media Group*

La prima settimana di Marzo si è svolta a Seattle (USA) la "Terza Conferenza Internazionale sui CD-ROM" organizzata dalla Microsoft.

Il titolo di questa terza conferenza era "Le industrie emergono", a sottolineare come la situazione stia evolvendo rapidamente e questa tecnologia si stia affermando.

Un anno fa i CD-ROM erano una tecnologia promettente, ora ci troviamo nel momento in cui molte società hanno iniziato a studiare e lavorare con questo nuovo oggetto e probabilmente l'anno prossimo sarà l'anno delle applicazioni.

A dimostrazione di questa tendenza il numero di partecipanti alla conferenza di quest'anno è stato notevolmente superiore a quello degli anni scorsi, si parla di circa 1300 persone. Anche il numero delle società presenti all'Exhibit che si é svolto durante la conferenza è risultato essere più del doppio di quello dello scorso anno.

Infine anche tra i relatori vi sono state personalità di notevole interesse come Bill Gates (Microsoft), John Sculley (Apple), Jim Manzi (Lotus), Joe Dionne (McGraw-Hill) e molti altri.

L'agenda della conferenza era così organizzata:

— la mattina della prima giornata è stata dedicata agli annunci da parte delle società più importanti, sono stati, per esempio, presentati i nuovi lettori di CD-ROM (Hitachi, Apple, Microcorp) ed e nuovi prodotti software della Microsoft;

— la mattina degli altri due giorni è stata invece dedicata alla descrizione di ciò che già esiste, a ciò che si sta facendo e alle tendenze future. Alcuni interventi sono stati dedicati al CD-I, che già aveva creato molto interesse alla prima conferenza del 1986, al DVI che era già stato presentato nel 1987 e del quale è stato mostrato un esempio. Si è parlato molto anche di hypertesto, e hypermedia, a questo proposito è da sottolineare il successo che ha ottenuto Hypercard presentato da Apple;

— il pomeriggio dei tre giorni è stato dedicato all'Exhibit, cioè all'esposizione (tipo EIKON) dei diversi prodotti da parte delle società presenti, e parallelamente a seminari su argomenti specifici, come per esempio l'andamento del mercato dei CD-ROM nel mondo. È da sottolineare a questo proposito che l'Italia é risultata essere il primo Paese in Europa per numero di lettori di CD-ROM venduti.

Finalmente sono arrivati, e sono stati presentati, i primi lettori di CD-ROM che permettono l'ascolto anche dei CD-Audio, personalmente ho potuto ascoltare la musica di un CD-Adio.

Tra molte novità questa conferenza ha segnato anche l'approvazione definitiva dello standard High Sierra per quanto riguarda la memorizzazione delle informazioni su CD-ROM, e quindi la possibilità per il futuro di realizzare CD-ROM che possono essere consultati da computer di marche diverse senza problemi.

Voci di corridoio hanno riferito che Philips e Sony starebbero studiando e definendo uno standard comune anche per i dischi WORM (Write Once Read Many times).

LE ULTIME NOVITÀ SULLE MEMORIE OTTICHE - *a cura di Media Group*

DVI (Digital Video Interactive).
Il DVI può essere considerato come lo strumento che apre le possibilità a nuove applicazioni sempre però legate alle memorie ottiche ed in particolare ai CD-ROM.
e Il DVI é stato sviluppato dalla General Electric e dalla RCA ed é il frutto di circa tre anni di studi e di lavoro.

La tecnologia del DVI combina le caratteristiche di interattività delle possibilità grafiche di un personal computer con la qualità grafica ed il realismo delle immagini in movimento dei video e con il suono, di tutto un ambiente digitale integrato. Il DVI é di mostrare un'ora di immagini in movimento memorizzate in forma digitale compressa e memorizzate su un CD-ROM standard.

La grafica DVI può anche essere combinata con altri oggetti sul video, testo, grafica e suono, tutto sempre controllato dell'utente.

L'utilizzo di questo oggetto non é attualmente alla portata di tutti poichè i suoi costi sono ancora molto alti. Infatti per poter memorizzare un'ora di video su un CD-ROM standard il DVI utilizza uno schema di Compressione e Decompressione delle informazioni molto complesso, ovviamente la compressione dei dati viene eseguita prima della masterizzazione, ed essendo un processo molto complesso e lungo richiede l'utilizzo di computer che ha una potenza di elaborazione molto superiore a quella di un normale personal.

Anche in fase di lettura del CD-ROM il DVI richiede un hardware specifico, e costoso, collegabile ad un personal (IBM PC AT) che permetta il trasferimento delle informazioni da CD-ROM a computer, la decompressione e la visualizzazione in tempi molto brevi così da poter riprodurre il movimento.

Al convegno EIKON e alla Terza Conferenza sui CD-

ROM tenutasi a Seattle è stato mostrato un esempio di "filmato" che utilizza il DVI e si é dimostrato un oggetto realmente molto interessante, soprattutto se riusciranno ad abbassare i costi.

L'esempio più significativo che illustra le possibilità di questo oggetto è stato mostrato all'EIKON e permette all'utente di immaginare di trovarsi davanti ad una piramide Azteca.

L'utente vede nello schermo del computer la piramide Azteca, ed ha a disposizione un joystick che può utilizzare per muoversi liberamente attorno ad essa, entrare, salire e guardarsi attorno variando la prospettiva della visione poichè in ogni istante egli vede nel monitor ciò che con i propri occhi vedrebbe se si trovasse fisicamente in quel luogo.

Ad ogni minimo movimento del joystick corrisponde la variazione dell'immagine visualizzata e l'effetto che è stato ottenuto è veramente molto interessante.

RECENSIONI

J.C. Simon, PATTERNS AND OPERATORS, THE FOUNDATION OF DATA REPRESENTATION, North Oxford Academic, Kogan Page, 1986, pp. 212

Si tratta di un testo che presenta una ampia introduzione agli elementi essenziali dei metodi algoritmici del pattern recognition, una disciplina che sta assumendo una sua fisionomia sempre più autonoma, pur utilizzando metodi e risultati di base di ricerche che coinvolgono tantissimi settori dell'informatica.

Questo testo è indirizzato verso un'utenza composta di studenti e ricercatori con competenze informatiche di base ma anche a chi, nel mondo della produzione, tiene un occhio alle discipline scientifiche e alle applicazioni d'avanguardia, ma il testo, in sè, può venire consigliato soprattutto a chi è interessato agli aspetti di carattere più teorico.

L'autore è Jean-Claude Simon, uno studioso di fama internazionale che presenta un approccio unificato agli algoritmi e al loro uso nella classificazione dei dati trattati con metodi informatici e alla loro rappresentazione. Tale approccio viene sommariamente esemplificato su un largo spettro di campi applicativi, inclusi la medicina, la geofisica e la teoria dei segnali.

Il testo è strutturato in sei capitoli principali, in cui nel primo vengono introdotti alcuni concetti base della teoria degli algoritmi, con un particolare riguardo ai problemi della complessità: ovviamente la panoramica dei più iportanti algoritmi presentati non ha pretese di esaustività, ma è impostata nell'ottica del problema dei segnali e del trattamento delle immagini.

Il secondo capitolo entra nello specifico del problema della rappresentazione e dell'interpretazione dell'informazione e dei principali metodi messi a punto per il pattern recognition, mentre il terzo capitolo tratta in particolare il problema della rappresentazione e del trattamento dei segnali evidenziandone i più noti metodi, dai frattali alle trasformate di Fourier, fino al trattamento di immagini con l'algoritmo di Karhunen-Loéve.

Nel quarto capitolo vengono prese in considerazione le strutture di dati e alcuni metodi di rappresentazione della conoscenza con i relativi modelli computazionali e il quinto capitolo è dedicato soprattutto alla rappresentazione e all'applicazione in termini di pattern recongnition della curva di Peano-Hilbert e a problemi di ricerca su grafi.

Infine, nel sesto capitolo viene illustrato lo spazio di rappresentazione euclidea e i metodi statistici impiegati nel pattern recognition e basati sulla teoria bayesiana.

La trattazione di tutti questi argomenti di fondamentale importanza è comunque essenziale e talvolta eccessivamente sintetica, ma una compensazione a tali caratteristiché è ovviata da un'ampia ed aggiornata bibliografia che correda il volume.

a cura di Stefania Bandini

Honeywell Bull N.SW.39/40 M011001 01

CIMINO MICHELE

H B ITALIA-DIREZ.GEN. MARKETING ITALIA

VIA PIRELLI,32
20124MILANO

Ufficio Distribuzione Documentazione Gestione Mailing List
Via del Parlamento, 33 - **20098 BORGOLOMBARDO (MI)**
Trattasi di trasporto di cancelleria-stampati-moduli-documenti interni-mobili e macchine per ufficio ESONERATI DALL'OBBLIGO DELLE BOLLE DI ACCOMPAGNAMENTO D.P.R. 6/10/78 n. 627 art. 4 n. 8 e Circolare Ministeriale n. 72 del 23/12/78 paragrafo 10